1fw

Docket No. 1232-5364

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s):	Toshiaki IIZUKA	Group Art Unit:	2122
Serial No.:	10/815,506	Examiner:	TBD
Filed:	March 31, 2004	Confirmation No.	1302

For: LOG ACQUISITION METHOD AND ITS CONTROL PROGRAM AND STORAGE MEDIUM

CERTIFICATE OF MAILING (37 C.F.R. §1.8(A))

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

I hereby certify that the attached:

1. Claim to Convention Priority
2. Certified Priority document - Japanese Patent Application Serial No. 2003-099465, filed April 2, 2003
3. Change of Correspondence Address
4. Return receipt postcard

along with any paper(s) referred to as being attached or enclosed and this Certificate of Mailing are being deposited with the United States Postal Service on date shown below with sufficient postage as first-class mail in an envelope addressed to the: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Respectfully submitted,
MORGAN & FINNEGAN, L.L.P.

Dated: June 2 2004

By: _____
Matthew K. Blackburn
Reg. No. 47,428

Correspondence Address:
MORGAN & FINNEGAN, L.L.P.
345 Park Avenue
New York, NY 10154-0053
(212) 758-4800 Telephone
(212) 751-6849 Facsimile

851606 v1

Docket No.: 1232-5364

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s):	Toshiaki IIZUKA	Group Art Unit	2122
Serial No.:	10/815,506	Examiner:	TBD
Filed:	March 31, 2004	Confirmation No.	1302

For: LOG ACQUISITION METHOD AND ITS CONTROL PROGRAM AND STORAGE MEDIUM

CLAIM TO CONVENTION PRIORITY

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

In the matter of the above-identified application and under the provisions of 35 U.S.C. §119 and 37 C.F.R. §1.55, applicant(s) claim(s) the benefit of the following prior application(s):

Application(s) filed in:	Japan
In the name of:	Canon Kabushiki Kaisha
Serial No(s):	2003-099465
Filing Date(s):	April 2, 2003

☒ Pursuant to the Claim to Priority, applicant(s) submit(s) a duly certified copy of said foreign application.

☐ A duly certified copy of said foreign application is in the file of application Serial No. _____, filed _____.

Respectfully submitted,
MORGAN & FINNEGAN, L.L.P.

Dated: ~~June~~ July 2 2004

By: _____
Matthew K. Blackburn
Registration No. 47,428

Correspondence Address:
MORGAN & FINNEGAN, L.L.P.
345 Park Avenue
New York, NY 10154-0053
(212) 758-4800 Telephone
(212) 751-6849 Facsimile

CFM03525
US
CN

日本国特許庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application: ２００３年　４月　２日

出願番号
Application Number: 特願２００３－０９９４６５

[ST. 10/C]: [ＪＰ２００３－０９９４６５]

出願人
Applicant(s): キヤノン株式会社

２００４年　４月１９日

特許庁長官
Commissioner,
Japan Patent Office 今井康夫

出証番号　出証特２００４－３０３２８６３

【書類名】　特許願
【整理番号】　253789
【提出日】　平成15年 4月 2日
【あて先】　特許庁長官殿
【国際特許分類】　G06F 9/00
【発明の名称】　ログ取得方法
【請求項の数】　1
【発明者】
【住所又は居所】　東京都大田区下丸子３丁目３０番２号　キヤノン株式会社内
【氏名】　飯塚　利明
【特許出願人】
【識別番号】　000001007
【氏名又は名称】　キヤノン株式会社
【代理人】
【識別番号】　100076428
【弁理士】
【氏名又は名称】　大塚　康徳
【電話番号】　03-5276-3241
【選任した代理人】
【識別番号】　100112508
【弁理士】
【氏名又は名称】　高柳　司郎
【電話番号】　03-5276-3241
【選任した代理人】
【識別番号】　100115071
【弁理士】
【氏名又は名称】　大塚　康弘
【電話番号】　03-5276-3241

【選任した代理人】

【識別番号】　100116894

【弁理士】

【氏名又は名称】　木村　秀二

【電話番号】　03-5276-3241

【手数料の表示】

【予納台帳番号】　003458

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【包括委任状番号】　0102485

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　ログ取得方法

【特許請求の範囲】

【請求項１】　所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に呼び出されるオペレーティングシステム内の関数のうち、指定された関数を識別する工程と、

ロードされた前記所定の処理を行う関数のアドレスと前記指定されたオペレーティングシステム内の関数のアドレスとを、ログ取得のための関数のアドレスに書き換える工程と、を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程と

を備えることを特徴とするログ取得方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、複数にモジュール分けされたソフトウェアの処理ログを取得するための技術に関するものである。

【０００２】

【従来の技術】

従来より、再現率の低いソフトウェアの障害に対しては、ソフトウェアの処理ログを取得し、当該処理ログを解析することによって、障害の原因をつきとめ、その対策を講じてきた（例えば、特許文献１を参照）。

【０００３】

【特許文献１】

特開平１１－２９６４１５号公報

【０００４】

【発明が解決しようとする課題】

しかし、上記従来の処理ログの取得には以下のような問題点がある。

（１）ユーザの動作環境でも処理ログを取得しつづけるためには、ソフトウェアのモジュール自体に手を加え、処理ログ取得ルーチンを追加しなければならず、処理ログ取得のための作業負荷が大きかった。

（２）処理ログ取得はモジュール毎に行うため、生成された処理ログはモジュール単位のものとなってしまい、ソフトウェア全体の処理を、完全に時間順の処理ログとして取得するのが困難である。このため、全体の処理ログとしての見通しが悪く、処理ログを解析して障害の原因を発見するまでのプロセスに工数がかかっていた。

【０００５】

本発明は、上記課題を鑑みてなされたものであり、複数にモジュール分けされたソフトウェアの処理ログを容易に取得でき、かつ、ソフトウェアの障害の原因の解析のための工数を削減することが可能なログ取得方法、ならびに該方法をコンピュータによって実現させるためのプログラム、および該プログラムを格納した記憶媒体を提供することを目的とする。

【０００６】

【課題を解決するための手段】

上記の目的を達成するために本発明に係るログ取得方法は以下のような構成を備える。即ち、

所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に呼び出されるオペレーティングシステム内の関数のうち、指定された関数を識別する工程と、

ロードされた前記所定の処理を行う関数のアドレスと前記指定されたオペレーティングシステム内の関数のアドレスとを、ログ取得のための関数のアドレスに

書き換える工程と、を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程とを備える。

【０００７】

【発明の実施の形態】

【第１の実施形態】

本実施形態は、あるモジュールから別のモジュール内に存在する関数の呼び出しが行われる際の仕組みである、メモリに保持されたインポート関数アドレス、または仮想関数アドレステーブル（Ｖｉｒｔｕａｌ　Ａｄｄｒｅｓｓ　Ｔａｂｌｅ）を利用して、モジュール間の関数呼び出しをフックして処理ログに記録することで、ソフトウェアのモジュール自体に手を加えることなく、ソフトウエア全体の処理を、時間順の処理ログとして取得することを可能にするものである。以下に具体的に説明する。

【０００８】

＜システム構成＞

図１は、本発明の各実施形態にかかるログ取得方法を実現するコンピュータ（ソフトウェア評価システム）の構成をあらわす図である。説明を簡略化するために、本実施形態では、本ソフトウェア評価システムが１台のＰＣ内部に構築されるものとするが、本発明にかかるログ取得方法は１台のＰＣ内部に構築されるか、あるいは複数のＰＣにネットワークシステムとして構築されるかによらず有効である。

【０００９】

本ログ取得方法を実現するソフトウェア評価システムには、ＣＰＵ１、チップセット２、ＲＡＭ３、ハードディスクコントローラ４、ディスプレイコントロー

ラ５、ハードディスクドライブ６、ＣＤ－ＲＯＭドライブ７、ディスプレイ８が搭載されている。また、ＣＰＵ１とチップセット２とを繋ぐ信号線１１、チップセット２とＲＡＭ３とを繋ぐ信号線１２、チップセット２と各種周辺機器とを繋ぐ周辺機器バス１３、ハードディスクコントローラ４とハードディスクドライブ６とを繋ぐ信号線１４、ハードディスクコントローラ４とＣＤ－ＲＯＭドライブ７とを繋ぐ信号線１５、ディスプレイコントローラ５とディスプレイ８とを繋ぐ信号線１６が搭載されている。

【００１０】

＜関数処理に対する処理ログ取得＞

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理を説明するために、まず図２によって、複数のモジュールに分かれたソフトウェアが、通常の状態でどのようにメモリにロードされるかを説明する。

【００１１】

通常、複数のモジュールに分かれたソフトウェアは、全体の制御を行う実行ファイルＥＸＥ（２３）と、モジュールとして存在しＥＸＥの補完的な役割を担うダイナミックリンクライブラリＤＬＬ（２７）とに分かれており、メモリにはＥＸＥとＤＬＬの両方がロードされる。ＥＸＥはコードセグメント（２８）とデータセグメント（２９）、そしてインポート関数アドレステーブル（２２）とから成っている。更に、インポート関数アドレステーブル（２２）は、関数の所属するＤＬＬによって分かれており（２１，２４）、ＤＬＬごとにそれぞれの関数がロードされたアドレスが書かれている（３０～３５）。ＤＬＬの関数の実体は、ＤＬＬごとに分かれて（２５，２６）ロードされ、それぞれの関数は該当するＤＬＬの一部としてロードされる（３６～４１）。この図では、１本のＥＸＥがＡ．ＤＬＬ及びＢ．ＤＬＬの２つのダイナミックリンクライブラリ内の関数を使用している例を示しており、実際に使用される関数はＦｕｎｃ　ＡＡ，　Ｆｕｎｃ　ＡＢ，　Ｆｕｎｃ　ＡＣ，　Ｆｕｎｃ　ＢＡ，　Ｆｕｎｃ　ＢＢ，　Ｆｕｎｃ　ＢＣの６個となっている。

【００１２】

ＥＸＥのコードセグメント内にあるコードが関数Ｆｕｎｃ　ＡＡを呼び出す場合には、まずインポート関数アドレステーブル（２２）内に書かれたＦｕｎｃ　ＡＡのアドレス（３０）が読み込まれる。ここには実際にはＡ．ＤＬＬの一部として読み込まれたＦｕｎｃ　ＡＡコード（３６）のアドレスが書かれており、そのアドレスをコールすることによって、ＥＸＥのコードはＡ．ＤＬＬのＦｕｎｃ　ＡＡを呼び出すことができる。

【００１３】

図３は、本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのメモリ構成をあらわす図であり、図２とは、ログ取得用のコードに対してＩＡＴ　Ｐａｔｃｈ（Ｉｍｐｏｒｔ　Ａｄｄｒｅｓｓ　Ｔａｂｌｅ　Ｐａｔｃｈ）という手法を用いて、関数呼び出しをリダイレクトしているという点で異なっている。

【００１４】

ログ取得が開始されると、メモリ内にはＩＡＴ　Ｐａｔｃｈ用のＤＬＬであるＣ．ＤＬＬ（５８）がロードされる。Ｃ．ＤＬＬ（５８）はインポート関数アドレステーブル（５２）内に書かれた関数のアドレスを、Ｃ．ＤＬＬ（５８）内のログ取得コードであるＦｕｎｃ　ＣＡＡ，Ｆｕｎｃ　ＣＡＢ，Ｆｕｎｃ　ＣＡＣ，Ｆｕｎｃ　ＣＢＡ，Ｆｕｎｃ　ＣＢＢ，Ｆｕｎｃ　ＣＢＣのアドレスに書き換える（６１～６６）。Ｃ．ＤＬＬ（５８）内のＦｕｎｃ　ＣＡＡ，Ｆｕｎｃ　ＣＡＢ，Ｆｕｎｃ　ＣＡＣ，Ｆｕｎｃ　ＣＢＡ，Ｆｕｎｃ　ＣＢＢ，Ｆｕｎｃ　ＣＢＣのコード（７３～７８）は、ログを記録すると共に、元々の関数呼び出しを受けるべくメモリにロードされている、該当する関数であるＦｕｎｃ　ＡＡ，Ｆｕｎｃ　ＡＢ，Ｆｕｎｃ　ＡＣ，Ｆｕｎｃ　ＢＡ，Ｆｕｎｃ　ＢＢ，Ｆｕｎｃ　ＢＣ（６７～７２）を呼び出す。

【００１５】

図４Ａは、図３におけるＩＡＴ　Ｐａｔｃｈの処理をあらわす図であり、図４Ｂはログ取得処理の流れを示すフローチャートである。説明を簡略化するために、この図ではＥＸＥがＡ．ＤＬＬ（５５）内のＦｕｎｃ　ＡＡを呼び出す際に、ＩＡＴ　Ｐａｔｃｈによるログ取得コードがどのように動作するかの例をあらわ

している。

【００１６】

ＥＸＥ（図４Ａの９１）がＦｕｎｃ　ＡＡをコールすると（図４Ａの９４）、Ｃ．ＤＬＬ（５８）内にあるログ取得コードがＤＬＬ名（Ｃ．ＤＬＬ）／関数名（Ｆｕｎｃ　ＡＡ）をメモリに保存し（図４Ｂのステップ Ｓ４０２）、呼び出し時刻をメモリに保存し、呼び出し時のパラメータをメモリに保存し、呼び出し時のポインタパラメータの指すメモリ内容を、別メモリに保存する（図４Ａの９５、図４ＢのステップＳ４０３）。その後Ｃ．ＤＬＬは本来呼び出されるはずであった、Ａ．ＤＬＬ（図４Ａの９３）内のＦｕｎｃ　ＡＡをコールする（図４Ａの９６、図４ＢのステップＳ４０４）。Ａ．ＤＬＬのＦｕｎｃ　ＡＡ処理（図４Ａの９７）が終了し、Ｃ．ＤＬＬに制御がリターンすると（図４Ａの９８）、Ｃ．ＤＬＬはリターン時の時刻をメモリに保存し、戻り値をメモリに保存し、リターン時にポインタパラメータが指すメモリ内容を、別メモリに保存する（図４Ａの９９）。その後、Ｃ．ＤＬＬは保存したログ情報をファイルに書き込み（図４Ａの１００、図４ＢのステップＳ４０５）、あたかもＡ．ＤＬＬのＦｕｎｃ　ＡＡが通常通りに終了したかのように、ＥＸＥにリターンする（１０１）。

【００１７】

図５は、本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの機能構成をあらわす図である。通常は実行形式のＥＸＥ（１１３）が、ＤＬＬ－１（１１６）やＤＬＬ－２（１１７）内の関数を呼び出すが、ここではＡＰＩトレーサと呼ばれるログ取得コードを埋め込み（１１４）、処理ログを生成している（１１５）。ＡＰＩトレーサは、ＤＬＬ－１やＤＬＬ－２の関数定義を記述したファイル（１１１）と、どのＤＬＬのどの関数のインポート関数テーブルを書き換えてログを取得するかの設定シナリオ（トレースシナリオ１１２）を元に動作する。

【００１８】

＜メソッド処理に対する処理ログ取得＞

次に、本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、実行ファイルＥＸＥがＣＯＭ（Ｃｏｍｐｏｎｅｎｔ　Ｏ

ｂｊｅｃｔ　Ｍｏｄｅｌ）サーバでエクスポートされているインターフェースのインスタンス作成時に、どのようにメモリにロードされるかを説明するために、まず、図６によって、通常の状態でどのようにメモリにロードされるかを説明する。

【００１９】

通常、インターフェースのインスタンス作成を行うと、ＣＯＭサーバ内で、要求されたインターフェース（１２１，１２２）と、そのメソッド（：オブジェクト指向プログラミングにおいて、オブジェクトの実行する手続きを記述したプログラム、１３０～１３５）が作成され、それらは、メモリ上に両方がロードされる。ここで、ｖｉｒｔｕａｌ　ａｄｄｒｅｓｓ　ｔａｂｌｅ（仮想アドレステーブル）は作成された各インターフェース毎に作られ（１１８，１２０）、作成要求を行ったＥＸＥに渡される。このｖｉｒｔｕａｌ　ａｄｄｒｅｓ　ｔａｂｌｅには各メソッドの作成されたアドレスが書かれている（１２４～１２９）。ＥＸＥはこれら情報を利用し、各インターフェースに対して呼び出しを行う。この図では、１本のＥＸＥがＩｎｔｅｒｆａｃｅ　Ａ及びＩｎｔｅｒｆａｃｅ　Ｂの２つのインターフェースのインスタンスを作成しており、そのインターフェース内部のメソッドを使用している例を示しており、実際に使用されているメソッドは、Ｍｅｔｈｏｄ　ＡＡ，Ｍｅｔｈｏｄ　ＡＢ，Ｍｅｔｈｏｄ　ＡＣ，Ｍｅｔｈｏｄ　ＢＡ，Ｍｅｔｈｏｄ　ＢＢ，Ｍｅｔｈｏｄ　ＢＣとなっている。

【００２０】

ＥＸＥのコードが関数Ｍｅｔｈｏｄ　ＡＡを呼び出す場合には、まずｖｉｒｔｕａｌ　ａｄｄｒｅｓｓ　ｔａｂｌｅ内に書かれたＭｅｔｈｏｄ　ＡＡのアドレス（１２４）が読み込まれる。ここには実際にはＣＯＭサーバのＩｎｔｅｒｆａｃｅ　Ａの一部として作成されたＭｅｔｈｏｄ　ＡＡコード（１３０）のアドレスが書かれており、そのアドレスをコールすることによって、ＥＸＥのコードはＩｎｔｅｒｆａｃｅ　ＡのＭｅｔｈｏｄ　ＡＡを呼び出すことができる。

【００２１】

図７は、本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのメモリ構成をあらわす図であり、図６とは、ログ取得用のコー

ドに対してＶＴａｂｌｅ　Ｐａｔｃｈ（ｖｉｒｔｕａｌ　ａｄｄｒｅｓｓ　ｔａｂｌｅ　Ｐａｔｃｈ）という手法を用いて、メソッド呼び出しをリダイレクトしているという点で異なっている。

【００２２】

ログ取得が開始されると、メモリ内にはＶＴａｂｌｅ　Ｐａｔｃｈ用のＤＬＬ（１４３）がロードされる。このＤＬＬはｖｉｒｔｕａｌ　ａｄｄｒｅｓｓ　ｔａｂｌｅ（１３６，１３８）内に書かれたメソッドのアドレスを、ＤＬＬ内のログ取得コードであるＭｅｔｈｏｄ　Ａ’Ａ，Ｍｅｔｈｏｄ　Ａ’Ｂ，Ｍｅｔｈｏｄ　Ａ’Ｃ，Ｍｅｔｈｏｄ　Ｂ’Ａ，Ｍｅｔｈｏｄ　Ｂ’Ｂ，Ｍｅｔｈｏｄ　Ｂ’Ｃのアドレスに書き換える（１４５～１５０）。ＤＬＬ内のＭｅｔｈｏｄ　Ａ’Ａ，Ｍｅｔｈｏｄ　Ａ’Ｂ，Ｍｅｔｈｏｄ　Ａ’Ｃ，Ｍｅｔｈｏｄ　Ｂ’Ａ，Ｍｅｔｈｏｄ　Ｂ’Ｂ，Ｍｅｔｈｏｄ　Ｂ’Ｃのコード（１５７～１６２）は、ログを記録すると共に、元々のメソッド呼び出しを受けるべくメモリにロードされている、該当するメソッドであるＭｅｔｈｏｄ　ＡＡ，Ｍｅｔｈｏｄ　ＡＢ，Ｍｅｔｈｏｄ　ＡＣ，Ｍｅｔｈｏｄ　ＢＡ，Ｍｅｔｈｏｄ　ＢＢ，Ｍｅｔｈｏｄ　ＢＣ（１５７～１６２）を呼び出す。

【００２３】

図８Ａは、図７におけるＶＴａｂｌｅ　Ｐａｔｃｈの処理をあらわす図、図８Ｂはログ取得処理の流れを示すフローチャートである。説明を簡略化するために、この図ではＥＸＥがＣＯＭサーバ内のＩｎｔｅｒｆａｃｅ　ＡのＭｅｔｈｏｄ　ＡＡを呼び出す際に、ＶＴａｂｌｅ　Ｐａｔｃｈによるログ取得コードがどのように動作するかの例をあらわしている。

【００２４】

ＥＸＥ（図８Ａの１６３）がＭｅｔｈｏｄ　ＡＡをコールすると（図８Ａの１６６）、ＤＬＬ内にあるログ取得コードがモジュール名／インターフェース名／メソッド名をメモリに保存し（図８Ｂのステップ８０２）、呼び出し時刻をメモリに保存し、呼び出し時のパラメータをメモリに保存し、呼び出し時のポインタパラメータの指すメモリ内容を、別メモリに保存する（図８Ａの１６７、図８Ｂのステップ８０３）。その後ＤＬＬは本来呼び出されるはずであった、ＣＯ

Mサーバ（図８Ａの１６５）内のＭｅｔｈｏｄ　ＡＡをコールする（図８Ａの１６８、図８ＢのステップＳ８０４）。ＣＯＭサーバのＭｅｔｈｏｄ　ＡＡ処理（図８Ａの１６９）が終了し、ＤＬＬに制御がリターンすると（図８Ａの１７０）、ＤＬＬはリターン時の時刻をメモリに保存し、戻り値をメモリに保存し、リターン時にポインタパラメータが指すメモリ内容を、別メモリに保存する（図８Ａの１７１）。その後、ＤＬＬは保存したログ情報をファイルに書き込み（図８Ａの１７２、図８ＢのステップＳ８０５）、あたかもＣＯＭサーバのＭｅｔｈｏｄ　ＡＡが通常通りに終了したかのように、ＥＸＥにリターンする（図８Ａの１７３）。

【００２５】

図９は、本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの機能構成をあらわす図である。通常は実行形式のＥＸＥ（１７６）が、ＣＯＭサーバ１（１７９）やＣＯＭサーバ２（１８０）内のメソッドを呼び出すが、ここではＡＰＩトレーサと呼ばれるログ取得コードを埋め込み（１７７）、処理ログを生成している（１７８）。ＡＰＩトレーサは、ＣＯＭサーバ１（１７９）やＣＯＭサーバ２の関数定義を記述したファイル（１７４）と、どのＣＯＭサーバのどのインターフェースのどのメソッドのｖｉｒｔｕａｌ　ａｄｄｒｅｓｓ　ｔａｂｌｅを書き換えてログを取得するかの設定シナリオ（１７５）を元に動作する。

【００２６】

＜実施例＞

図１０は、本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムに対して、それぞれの関数及びメソッドのパラメータの形式や、戻り値の形式を指示する、関数定義ファイルの例を示す図である。ＤＬＬ／インターフェース名及び関数／メソッド名を記述し（「関数／メソッド」とは、「関数またはメソッド」の意、以下同じ）、その関数／メソッドに対する、パラメータ及び戻り値の型が示されている。本実施形態にかかるログ取得方法を実現するソフトウェア評価システムは、この関数定義ファイルによって指示された内容を元に、それぞれの関数／メソッドがどのようなパラメータ／戻り値を有してい

るかを判断し、その内容をログとして取得する。

【００２７】

図１１は、図１０に示した関数定義ファイルを用いて、本発明の実施形態にかかるソフトウェア評価システムで取得した、ログの一例を示す図である。それぞれの呼び出しに対して、関数／メソッドが呼び出された時刻（Ｉｎ時刻、Ｏｕｔ時刻）、及びその際のパラメータ／戻り値が、ログとして生成される。

【００２８】

以上の説明から明らかなように、本実施形態にかかるログ取得方法によれば、複数にモジュール分けされたソフトウエアの処理ログの取得において、モジュール自体に変更を加えることなく、モジュール内に用意された関数／メソッドの呼び出しを処理ログとして記録することが可能となり、処理ログ取得のための作業負荷を低減することが可能となる。また、生成される処理ログは、時間順の処理ログとして取得することができ、処理ログの解析が容易となるため、ソフトウェアの障害の原因の解析のための工数を削減することが可能となる。

【００２９】

【第２の実施形態（その１）】

上記第１の実施形態では、ＥＸＥによって呼び出される関数／メソッドすべてについて処理ログを取得することとした。しかしながら、この場合、取得される処理ログの数が膨大になってしまうことも考えられる。そこで、本実施形態ではＥＸＥによって呼び出される関数／メソッドのうち、処理ログの取得対象を限定する場合について述べる。

【００３０】

一般にソフトウェアのモジュール構成は、ＯＳ　Ｓｕｒｆａｃｅによって ＯＳ部分のＤＬＬとアプリケーション部分のＤＬＬとに分けられる。図１２は、一般的なソフトウェアのモジュール構成の一例を示す図である。同図において、１２００はＡＰＰ．ＥＸＥであり、１２０１～１２０３はそれぞれ、ＭｏｄｕｌｅＡ．ｄｌｌ、ＭｏｄｕｌｅＢ．ｄｌｌ、ＭｏｄｕｌｅＣ．ｄｌｌであり、アプリケーション部分のＤＬＬである。一方、１２０５～１２１０はそれぞれ、Ｕｓｅｒ３２．ｄｌｌ、ＧＤＩ３２．ｄｌｌ、Ｎｔｄｌｌ．ｄｌｌ、Ｗｓ２＿３２．ｄｌ

ｌ、Ｕｎｉｄｒｖ．ｄｌｌであり、ＯＳ部分のＤＬＬである。

【００３１】

そこで、本実施形態では、このようなモジュール構成を備えるソフトウェアにおいて、アプリケーション部分のＤＬＬ内の関数が呼び出された場合のみ処理ログを取得することで処理ログの収集対象を限定する。

【００３２】

図１３は、オペレーティング・システムのモジュールを定義する例を示す図である。このモジュール定義をもって、図１４に示した処理が行なわれる。

【００３３】

図１４は、本発明の第２の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

【００３４】

ステップＳ１４００で処理が開始されると、まず、図１３で示したオペレーティング・システムのモジュール定義１３００を取得し、ＯＳモジュールリストを保持する（ステップＳ１４０１）。続いて、関数／メソッドの呼び出し元のモジュールが、そのＯＳモジュールリストに存在するか否かを判別し（ステップＳ１４０２、Ｓ１４０３）、存在する場合は呼び出し先のモジュールをＯＳモジュールリストに追加する（ステップＳ１４０９）だけで、そのまま元の関数／メソッドを呼び出す（ステップＳ１４１０）。このステップＳ１４０９の処理により、更にそのモジュール以下の呼び出しが行われた場合でも、その呼び出しの処理ログを取得しない手段が確立されている。

【００３５】

一方、ステップＳ１４０３においてＯＳモジュールリストに存在しない場合は、ＤＬＬ名／インターフェース名／関数名／メソッド名／呼び出し時の時刻をＨＤＤに保存し（ステップＳ１４０４）、その呼び出しに対してのパラメータをＨＤＤに保存する（ステップＳ１４０５）。

【００３６】

続いて、本体の関数／メソッドを呼び出す（ステップＳ１４０６）。関数／メソッド内部の処理が終了すると、ＤＬＬ名／インターフェース名／関数名／メソ

ッド名／終了時の時刻をＨＤＤに保存し（ステップＳ１４０７）、その呼び出しに対してのパラメータ及び戻り値をＨＤＤに保存する（ステップＳ１４０８）。続いて、関数／メソッドのリターン処理を行なう（ステップＳ１４１１）。この処理は、評価対象となるプログラムが終了するまで続行される（ステップＳ１４１２）。

【００３７】

図１５Ａは、図１４における処理によって取得された処理ログの例である。なお、比較のために図１５ＢにＯＳ部分のＤＬＬ内の関数が呼び出された際にも処理ログを取得することとした場合の例を示す。図１５Ａと図１５Ｂとを対比することで明らかなように、図１３におけるオペレーティング・システムのモジュール定義１３００により、図１５ＡではＫｅｒｎｅｌ３２．ｄｌｌからＮｔｄｌｌ．ｄｌｌに対する呼び出し、あるいはＵｓｅｒ３２．ｄｌｌからＷｓ２＿３２．ｄｌｌに対する呼び出しといった、オペレーティング・システムの内部の処理ログも含めた処理ログが取得されていないため、よりサイズの小さい処理ログで解析することが可能となっている。

【００３８】

以上の説明から明らかなように、本実施形態によれば、関数定義ファイルにおいて「オペレーティング・システムのサーフェイス」を指定することで、ソフトウェアに対する適切な処理ログが取得可能となり、バグ原因特定などの解析が可能となる。

【００３９】

【第２の実施形態（その２）】

上記第２の実施形態（その１）においては、オペレーティング・システムのサーフェイス以下すべての処理ログを取得しない例を挙げたが、実際はサーフェイス以下の一部（ＯＳ部分のＤＬＬ）に関しても、処理ログを取得したい場合がある。

【００４０】

例えば、上記第１の実施形態において示したログ取得方法を実現するためのプログラム（以下、「ログ取得プログラム」と称す）は、ドライバＳＤＫ（ＳＤＫ

＝Ｓｏｆｔｗａｒｅ　Ｄｅｖｅｌｏｐｍｅｎｔ　Ｋｉｔ）に組み込んで利用する場合ことが考えられる。

【００４１】

一般にドライバＳＤＫ（例えばプリンタＳＤＫ）は周辺機器メーカ（プリンタメーカ）により提供されるが、ドライバＳＤＫを用いるソフトウェア（例えば、プリンタに印刷する文書や画像を生成、編集するためのソフトウェア）は他のメーカが作成することが多い。このため、ソフトウェアがドライバＳＤＫ（プリンタＳＤＫ）を使用した際に正常に動作しなかった時は、その原因がドライバＳＤＫ（プリンタＳＤＫ）側のバグによるものなのか、ソフトウェア側のバグによるものなのかを判別することが重要となってくる。このようなケースにおいてはログ取得プログラムにより、プリンタＳＤＫの処理内容を処理ログとして取得しておくことが重要となってくる。そのためには、Ｕｎｉｄｒｖ．ｄｌｌ（１２１０）などのＯＳ部分のＤＬＬ内の関数を呼び出した場合の処理ログも収集対象としておく必要がある。

【００４２】

つまり、処理ログの取得対象を限定するにあたっては、ログ取得プログラムの使用目的に応じて、任意に選択できるようにしておくことが望ましい。本実施形態では、かかる点に考慮したログ取得プログラムについて説明する。

【００４３】

図１６は、このような必要性に鑑みて、オペレーティング・システムの除外モジュール（ログ取得対象外として定義されるオペレーティング・システムのモジュールから除外するモジュール、つまり、ログ取得対象となるモジュール）を定義する例を示す図である。この除外モジュール定義及び図１３に示したサーフェイス定義をもって、図１７にあらわした処理が行なわれる。本実施形態では、サーフェイスから除外の指定をユーザが行なうことにより、上記のケースにおいてドライバの処理ログをソフトウェアの処理ログと併せて取得する。

【００４４】

図１７は、本発明の第２の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

【００４５】

ステップＳ１７００において処理が開始されると、まず本ログ取得プログラムは、図１３で示したオペレーティング・システムのモジュール定義１３００を取得し、ＯＳモジュールリストを保持する（ステップＳ１７０１）。続いて、図１６で示した除外モジュール定義を取得し、除外モジュールリストを保持する（ステップＳ１７０２）。続いて、関数／メソッドの呼び出し元のモジュールが存在するかどうかを、ＯＳモジュールリスト及び除外モジュールリストの両リストから検索する（ステップＳ１７０３）。

【００４６】

続いて、呼び出し元のモジュールがＯＳモジュールリストに存在するか否かを判別し（ステップＳ１７０４）、存在する場合は、呼び出し元が除外モジュールであるかどうかを判別する（ステップＳ１７０５）。ＯＳモジュールリストに存在し、且つ除外モジュールリストに存在しない場合、本ログ取得プログラムは、呼び出し先のモジュールをＯＳモジュールリストに追加する（ステップＳ１７１１）だけで、そのまま元の関数／メソッドを呼び出す（ステップＳ１７１２）。

【００４７】

呼び出し元のモジュールが、ステップＳ１７０４においてＯＳモジュールリストに存在しない場合、及びステップＳ１７０５において除外モジュールリストに存在する場合は、ＤＬＬ名／インターフェース名／関数名／メソッド名／呼び出し時の時刻をＨＤＤに保存し（ステップＳ１７０６）、その呼び出しに対してのパラメータをＨＤＤに保存する（ステップＳ１７０７）。

【００４８】

続いて、本体の関数／メソッドを呼び出す（ステップＳ１７０８）。関数／メソッド内部の処理が終了すると、ＤＬＬ名／インターフェース名／関数名／メソッド名／終了時の時刻をＨＤＤに保存し（ステップＳ１７０９）、その呼び出しに対してのパラメータ及び戻り値をＨＤＤに保存する（ステップＳ１７１０）。続いて、関数／メソッドのリターン処理を行なう（ステップＳ１７１３）。

【００４９】

この処理は、評価対象となるプログラムが終了するまで（ステップＳ１７１４

）続行される。

【００５０】

このように「オペレーティング・システムのサーフェイス」及び「サーフェイスからの除外モジュール」を指定可能なＡＰＩトレーサにより、ソフトウェアに対する適切な処理ログが取得可能となり、バグ原因特定などの解析が可能となる。

【００５１】

【第２の実施形態（その３）】

上記第２の実施形態（その１）、（その２）において、図１４のステップＳ１４０１、図１７のステップＳ１７０１では、ＯＳモジュールリストの取得をオペレーティング・システムが提供した情報によって自動的に取得することとしたがこれに限られない。また、「オペレーティング・システムが提供した情報」は、オペレーティング・システムが用意したインターフェースから取得するものでも構わないし、例えばＷｉｎｄｏｗｓ（登録商標）　ＯＳの場合、レジストリ情報でも構わない。

【００５２】

また、図１４に示した処理を行なうか、図１７に示した処理を行なうかを、ユーザが明示的に選択できるようにしてもよい。。

【００５３】

さらに、上記第２の実施形態（その２）では、あらかじめオペレーティング・システムの除外モジュールの定義がなされていることとしたが、図１７のステップＳ１７０２において、ユーザが任意に指定できるようにしてもよい。

【００５４】

【第３の実施形態（その１）】

上記第１の実施形態では、関数定義ファイルとして、ＤＬＬ名／インターフェース名／関数名／メソッド名と、その関数／メソッドに対するパラメータ及び戻り値の型とを記述することとしたが（図１０参照）、本発明にかかるログ取得方法を実現するソフトウェア評価システムにおける関数定義ファイルの記述は、これに限られない。本実施形態では、特殊な構造体データを含む関数として、ＤＥ

ＶＭＯＤＥを含む関数のログを取得する場合の、関数定義ファイルの記述方法と当該関数定義ファイルにおけるログ取得方法とについて説明する。

【００５５】

ＤＥＶＭＯＤＥとは、プリンタドライバのユーザインタフェースで設定できる印刷設定がどのように設定されているかを示すＷｉｎｄｏｗｓ（登録商標）の構造体のことをいい、ＤＥＶＭＯＤＥにはＯＳが共通で定義している部分と、プリンタベンダが拡張して使用できる部分（拡張領域）とがある。そして、ＤＥＶＭＯＤＥの最初に「拡張領域サイズ」が指定されている。このように拡張領域のサイズが指定されている場合には、ＤＥＶＭＯＤＥ構造体へのポインタがＡＰＩトレーサのパラメータとして入力された場合、ＤＥＶＭＯＤＥの標準領域（サイズ固定）の最後のアドレスから指定サイズ分だけデータ内容を読み出してログとして取得するように、関数定義ファイルに書き込むことで、ＤＥＶＭＯＤＥを含む関数の処理ログを取得することが可能となる。以下に詳細に説明する。

【００５６】

図１８は、本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義ファイルの一例であり、一般的に広く用いられているＩＤＬにより記述されている。本ソフトウェア評価システムに於いては、このＩＤＬをトークン化したタイプライブラリファイルを関数定義ファイルとして使用する。なお、本実施形態は付加バイナリ情報取得のための定義手段を提供することを特徴とするものであり、関数定義ファイルはＩＤＬでなくても構わない。例えば、通常のテキストファイルを用いることも可能であり、ＸＭＬなどのマークアップ言語を用いることも可能である。

【００５７】

図１９は、本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義ファイルを示す図であり、構造体へのポインタに対して付加バイナリ情報取得を指定することで、ポインタ・データの実体を処理ログとして取得すべく、ＩＤＬにより記述されている。

【００５８】

ＴＥＳＴＳＴＲＵＣＴ＿ＷＩＴＨＥＸＴＲＡ構造体の定義に於いて、ｌｏｎｇ

ｃＥｘｔｒａＢｉｎａｒｙＤａｔａに対してｃｕｓｔｕｍ（ＰＡＴ＿ＰＡＲＡＭ＿ＡＴＴＲ＿ＩＤ，”ｓｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓ（）”）と宣言している（１９０１）。ここで、ＰＡＴ＿ＰＡＲＡＭ＿ＡＴＴＲ＿ＩＤ（１９００）はＩＤＬの中で本ソフトウェア評価システムが利用するための識別子である。ここで、”ｓｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓ（）”と定義しておき、ＦｕｎｃＳｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓ関数に引数をＴＥＳＴＳＴＲＵＣＴ＿ＷＩＴＨＥＸＴＲＡ＊　ｌｐＰａｒａｍと定義しておく（１９０２）ことにより、ＦｕｎｃＳｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓ関数が実際に呼び出された際に、ＴＥＳＴＳＴＲＵＣＴ＿ＷＩＴＨＥＸＴＲＡ構造体の置かれたメモリ領域の先頭から、ｃＥｘｔｒａＢｉｎａｒｙＤａｔａが指すサイズ分だけの、付加情報としてのバイナリデータを処理ログとして保存する。

【００５９】

図２０は、図１９に示された関数定義による構造体が、どのようにメモリ上に配置されるかを表す図である。

【００６０】

この図では、ＴＥＳＴＳＴＲＵＣＴ＿ＷＩＴＨＥＸＴＲＡ構造体（２０００）が置かれたメモリの先頭アドレスの０ｘ０１Ｆ７Ｂ０００であるとして説明する。１ｂｙｔｅのデータであるｃｈＰａｒａｍ１（２００１）は０ｘ０１Ｆ７Ｂ００に置かれ、４ｂｙｔｅのデータであるｃＥｘｔｒａＢｉｎａｒｙＤａｔａＳｉｚｅ（２００２）は０ｘ０１Ｆ７Ｂ００１に置かれ、１ｂｙｔｅのデータであるｃｈＰａｒａｍ２（２００３）は０ｘ０１Ｆ７Ｂ００５に置かれ、１ｂｙｔｅのデータであるｃｈＰａｒａｍ３（２００４）は０ｘ０１Ｆ７Ｂ００６に置かれている。

【００６１】

ここまでは通常の構造体メモリ配置と同一であるが、ＴＥＳＴＳＴＲＵＣＴ＿ＷＩＴＨＥＸＴＲＡ構造体はｃｈＰａｒａｍ３に続くメモリ、すなわち０ｘ０１Ｆ７Ｂ００７からの領域にＥｘｔｒａＢｉｎａｒｙＤａｔａ（２００５）という付加データを持っており、そのサイズがｃＥｘｔｒａＢｉｎａｒｙＤａｔａＳｉｚｅに応じて可変となっている。例えば、ｃＥｘｔｒａＢｉｎａｒｙＤａｔａＳ

ｉｚｅの値として８が入っている場合には、ＥｘｔｒａＢｉｎａｒｙＤａｔａは８ｂｙｔｅ存在することになるため、ＴＥＳＴＳＴＲＵＣＴ＿ＷＩＴＨＥＸＴＲＡ構造体の最後尾のメモリアドレスは０ｘ０１Ｆ７Ｂ００Ｅとなる。このため、ＴＥＳＴＳＴＲＵＣＴ＿ＷＩＴＨＥＸＴＲＡ構造体の指し示すすべてのデータをログに取得するためには、このＥｘｔｒａＢｉｎａｒｙＤａｔａを、ｃＥｘｔｒａＢｉｎａｒｙＤａｔａＳｉｚｅの値の分だけバイナリデータとして取得する必要がある。図１９におけるｓｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓの定義は、この必要を満たすために記述されている。

【００６２】

図２１は、本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１９に示すように関数定義ファイルが記述されている時の処理ログを取得する場合の処理の流れを示すフローチャートである。

【００６３】

処理が開始されると（ステップＳ２１００）、ログ取得を開始し、モジュール名／インターフェース名／関数名／メソッド名をＨＤＤに保存する（ステップＳ２１０１）。次にログ取得コードは、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をＨＤＤに保存する（ステップＳ２１０２）。そして、関数定義にｓｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓの設定が存在するかどうか、すなわちｓｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓの定義を含んだ構造体がパラメータに存在するか否かを判別し（ステップＳ２１０３）、存在する場合には、まずその定義を含む構造体のサイズを計算する（ステップＳ２１０４）。

【００６４】

続いて、構造体の先頭ポインタから計算した構造体サイズの分だけずれたメモリ領域から、ｓｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓで示されたメンバの値によって示されるサイズ分だけ、データをＨＤＤに保存する（ステップＳ２１０５）。続いて、元の関数／メソッドを呼び出す（ステップＳ２１０６）。

【００６５】

関数／メソッドからリターンすると、ログ取得コードはリターン時の時刻、戻

り値及びポインタ・パラメータの指すメモリの内容をＨＤＤに保存する（ステップＳ２１０７）。そして、関数定義にｓｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓの設定が存在するかどうか、すなわちｓｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓの定義を含んだ構造体がパラメータ／戻り値に存在するか判別し（ステップＳ２１０８）、存在する場合には、その定義を含む構造体のサイズを計算する（ステップＳ２１０９）。続いて、構造体の先頭ポインタから計算した構造体サイズの分だけずれたメモリ領域から、ｓｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓで示されたメンバの値によって示されるサイズ分だけ、データをＨＤＤに保存する（ステップＳ２１１０）。この処理はユーザから終了の命令により終了する（ステップＳ２１１１）。なお、ｓｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓで指定されたメンバが複数にわたる場合は、当然ステップＳ２１０４～ステップＳ２１０５及びステップＳ２１０９～ステップＳ２１１０の処理が指定された数すべてに対して行なわれる。

【００６６】

図２２は、本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１９の関数定義ファイルにより取得された処理ログを示す図である。

【００６７】

ここでは処理ログはテキスト部分（２２００）とバイナリ部分（２２０１）とに分けて表示しており、テキストデータに書き込まれたＤａｔａＩＤが、バイナリ部分に書き込まれたＤａｔａＩＤに対応している。例えば、一度目の呼び出しのログではＤａｔａＩＤが０ｘ０００１であるが、このＩＤがバイナリ部分に書き込まれた付加バイナリ情報、ＤａｔａＩＤ=０ｘ０００１を指し示している。この場合、ＦｕｎｃＳｔｒｕｃｔｓｉｚｅｅｘｔｒａＩｓの１度目の呼び出しでは８バイト、２度目の呼び出しでは４０バイト、３度目の呼び出しでは５バイト、４度目の呼び出しでは７バイトの付加バイナリデータが取得されている。ｓｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓの定義を行なわない場合には、ｃｈＰａｒａｍ１及びｃＥｘｔｒａＢｉｎａｒｙＤａｔａＳｉｚｅ及びｃｈＰａｒａｍ２及びｃｈＰａｒａｍ３のみが取得されるが、ｓｔｒｕｃｔｓｉｚｅｅｘｔｒａ＿ｉｓ

の定義を行なうことにより、付加バイナリデータであるＥｘｔｒａＢｉｎａｒｙＤａｔａを、それぞれの呼び出しに応じたサイズ分、処理ログに取得することが可能となっている。

【００６８】

このように、構造体のメンバに応じた処理ログ取得のための関数定義手段を設けることにより、通常では取得しきれないデータを、バイナリデータとして処理ログに取得することが可能となる。

【００６９】

【第３の実施形態（その２）】

上記第３の実施形態（その１）では、特定の構造体の最後尾のアドレスから所定サイズ分のデータを処理ログとして取得するための関数定義ファイルの記述方法と、その場合のログ取得プログラムの処理について述べたが、本発明にかかるログ取得方法を実現するソフトウェア評価システムにおける関数定義ファイルの記述方法はこれに限られない。

【００７０】

図２３は、本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義ファイルを示す図であり、構造体へのポインタに対してメンバをオフセットとして指定することで、ポインタ・データの実体をログとして取得すべく、ＩＤＬにより記述されている。

【００７１】

ＴＥＳＴＳＴＲＵＣＴ＿ＷＩＴＨＰＴＲＤＩＦＦ構造体の定義に於いて、ＬＰＳＴＲ ｌｐｓｚＳｔｒｉｎｇ及びｌｏｎｇ ｎＮｕｍｂｅｒに対してｃｕｓｔｕｍ（ＰＡＴ＿ＰＡＲＡＭ＿ＡＴＴＲ＿ＩＤ，”ｐｔｒｄｉｆｆ　()”）と宣言している（２３０１、２３０２）。ここで、ＰＡＴ＿ＰＡＲＡＭ＿ＡＴＴＲ＿ＩＤ（２３００）はＩＤＬの中で本ソフトウェア評価システムが利用するための識別子である。ここで、”ｐｔｒｄｉｆｆ　()”と定義しておき、ＦｕｎｃＰｔｒｄｉｆｆ関数に引数をＴＥＳＴＳＴＲＵＣＴ＿ＷＩＴＨＰＴＲＤＩＦＦ＊ ｌｐＰａｒａｍと定義しておく（２３０３）ことにより、ＦｕｎｃＰｔｒｄｉｆｆ関数が実際に呼び出された際に、ＴＥＳＴＳＴＲＵＣＴ＿ＷＩＴＨＰＴＲＤＩＦＦ構

造体の置かれたメモリ領域の先頭から、ｌｐｓｚＳｔｒｉｎｇ及びｎＮｕｍｂｅｒの数値によって表されるオフセットだけずれたメモリ領域に格納された、ＬＰＳＴＲ及びｌｏｎｇのデータを処理ログとして保存する。

【００７２】

ｐｔｒｄｉｆｆ（）の宣言は、以下の２つの条件定義を兼ねている。

（１）ｐｔｒｄｉｆｆ（）で指定された引数は、実際には規定された型ではなく、構造体の先頭アドレスに対するオフセットを表す数値として扱われること。

（２）そのオフセットの指し示すメモリエリアに、規定された型のデータが保持されていること。

【００７３】

図２４は、図１８に示された関数定義による構造体が、どのようにメモリ上に配置されるかを表す図である。

【００７４】

この図では、ＴＥＳＴＳＴＲＵＣＴ＿ＷＩＴＨＰＴＲＤＩＦＦ構造体（２４００）が置かれたメモリの先頭アドレスの０ｘ０１Ｆ７Ｂ０００であるとして説明する。１ｂｙｔｅのデータであるｃｈＰａｒａｍ１（２４０１）は０ｘ０１Ｆ７Ｂ０００に置かれ、４ｂｙｔｅのデータであるｌｐｓｚＳｔｒｉｎｇへのオフセット（２４０２）は０ｘ０１Ｆ７Ｂ００１に置かれ、４ｂｙｔｅのデータであるｎＮｕｍｂｅｒへのオフセット（２４０３）は０ｘ０１Ｆ７Ｂ００５に置かれ、１ｂｙｔｅのデータであるｃｈＰａｒａｍ２（２４０４）は０ｘ０１Ｆ７Ｂ００９に置かれている。ＴＥＳＴＳＴＲＵＣＴ＿ＷＩＴＨＰＴＲＤＩＦＦ構造体において、ｌｐｓｚＳｔｒｉｎｇの実体はオフセット（２４０２）によって示されたサイズ分、構造体の先頭アドレスからずらした領域に格納されており（２４０５）、ｎＮｕｍｂｅｒの実体はオフセット（２４０３）によって示されたサイズ分、構造体の先頭アドレスからずらした領域に格納されている（２４０６）。このため、ＴＥＳＴＳＴＲＵＣＴ＿ＷＩＴＨＰＴＲＤＩＦＦ構造体の指し示すすべてのデータを処理ログに取得するためには、これらｌｐｓｚＳｔｒｉｎｇ及びｎＮｕｍｂｅｒに関して、実体のデータをログとして取得する必要がある。図２３におけるｐｔｒｄｉｆｆの定義は、この必要を満たすために記述されている。

【００７５】

図２５は、本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図２３に示すように関数が定義されている時のログを取得する場合の処理の流れを示すフローチャートである。

【００７６】

処理が開始されると（ステップＳ２５０１）、ログ取得を開始し、モジュール名／インターフェース名／関数名／メソッド名をＨＤＤに保存する（ステップＳ２５０２）。次にログ取得コードは、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をＨＤＤに保存する（ステップＳ２５０３）。そして、関数定義ファイルにｐｔｒｄｉｆｆの設定が存在するかどうか、すなわちｐｔｒｄｉｆｆの定義を含んだ構造体がパラメータに存在するか否かを判別し（ステップＳ２５０４）、存在する場合には、構造体の先頭アドレスとｐｔｒｄｉｆｆで定義された構造体メンバのオフセットの値を合算し、ｐｔｒｄｉｆｆで定義された構造体メンバがどのアドレスに格納されているかを計算する（ステップＳ２５０５）。続いて、計算したアドレスの指し示すメモリ領域から、ｐｔｒｄｉｆｆで示されたメンバの型のデータをＨＤＤに保存する（ステップＳ２５０６）。続いて、元の関数／メソッドを呼び出す（ステップＳ２５０７）。関数／メソッドからリターンすると、ログ取得コードはリターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内容をＨＤＤに保存する（ステップＳ２５０８）。そして、関数定義ファイルにｐｔｒｄｉｆｆの設定が存在するかどうか、すなわちｐｔｒｄｉｆｆの定義を含んだ構造体がパラメータ／戻り値に存在するか否かを判別し（ステップＳ２５０９）、存在する場合には、構造体の先頭アドレスとｐｔｒｄｉｆｆで定義された構造体メンバのオフセットの値を合算し、ｐｔｒｄｉｆｆで定義された構造体メンバがどのアドレスに格納されているかを計算する（ステップＳ２５１０）。続いて、計算したアドレスの指し示すメモリ領域から、ｐｔｒｄｉｆｆで示されたメンバの型のデータをＨＤＤに保存する（ステップＳ２５１１）。この処理はユーザから終了の命令により処理を終了する（ステップＳ２５１２）。なお、ｐｔｒｄｉｆｆで指定されたメンバが複数に渡る場合は、当然ステップＳ２５０５～ステップＳ２５０６及びステップＳ２５

１０～ステップＳ２５１１の処理が指定された数すべてに対して行なわれる。

【００７７】

図２６は、本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図２３の関数定義ファイルにより取得された処理ログを示す図である。それぞれの呼び出しでオフセットが異なっているが、本ソフトウェア評価システムは、オフセットで指し示された実体を的確に処理ログとして取得している。

【００７８】

以上の説明から明らかなように、本実施形態によれば、構造体のメンバに応じたログデータ取得のための定義手段を設けることにより、通常では取得しきれないデータを含む関数／メソッドであっても、処理ログを取得することが可能となる。

【００７９】

【第４の実施形態（その１）】

上記第１の実施形態では、関数定義ファイルとして、ＤＬＬ名／インターフェース名／関数名／メソッド名と、その関数／メソッドに対するパラメータ及び戻り値の型とを記述することとしたが（図１０参照）、本発明にかかるログ取得方法を実現するソフトウェア評価システムにける関数定義ファイルの記述は、これに限られない。本実施形態では、特殊なＣＯＭに対してログを取得する場合の、関数定義ファイルの記述方法について説明する。

【００８０】

一般に、プリンタドライバには、ＯＳに予め組み込まれているＵＮＩＤＲＩＶＥＲ（ユニドライバ）を用いる場合と、モノレシックドライバを用いる場合とがある。そして、プリンタベンダーはモノレシックドライバ、またはユニドライバからの出力を受け取るプラグインモジュール（＊．ｄｒｖ）を提供している。モノレシックドライバの場合は、ＧＤＩ３２．ｄｌｌは一般的なＤＤＩ関数を用いる。

【００８１】

ここで、ＤＬＬ同士は、レジストリに登録されているＡＰＩを用いてデータの

やり取りを行うが、ユニドライバがプリンタベンダが提供するプラグインモジュールに出力する処理内容は、レジストリに登録されていないＣＯＭ（ｏｂｊｅｃｔ－ＡＰＩ）である。このため、上記第１の実施形態において示した関数定義ファイルでは、特殊なＣＯＭについての処理ログを取得することができない。そこで、本実施形態ではユニドライバが吐き出すレジストリに登録されていないＣＯＭを通常のＣＯＭとして扱うように関数定義ファイルを記述することで適切なログを取得することとする。

【００８２】

図２７は、本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、それぞれの関数／メソッドのパラメータの形式や、戻り値の形式を指示する関数定義ファイルの例であり、一般的に広く用いられているＩＤＬにより記述されている。ＤＬＬ名／インターフェース名／関数名／メソッド名を記述し、その関数／メソッドに対する、パラメータ／戻り値の型が示されている。本発明の第４の実施形態にかかるソフトウェア評価システムに於いては、このＩＤＬをトークン化したタイプライブラリファイルを関数定義ファイルとして使用する。なお、本実施形態は、より詳細な情報取得のための関数定義手段を提供することを特徴とするものであり、関数定義ファイルはＩＤＬにより記述されていなくても構わない。例えば、通常のテキストファイルを用いることも可能であり、ＸＭＬなどのマークアップ言語を用いることも可能である。

【００８３】

本発明の第４の実施形態にかかるソフトウェア評価システムは、この関数定義ファイルによって指示された内容を元に、それぞれの関数／メソッドがどのようなパラメータ／戻り値を有しているかを判断し、その内容をログとして取得する。なお、本実施形態では、Ｉｎｔｅｒｆａｃｅ　ｔｅｓｔに実装されたＤｌｌＧｅｔＣｌａｓｓＯｂｊｅｃｔという名前の関数が、以下のようなパラメータを持つ状態を例としている。

（１）入力パラメータの”ｒｃｌｓｉｄ”、すなわち「クラスＩＤ」をあらわすパラメータ。

（２）入力パラメータの”ｒｉｉｄ”、すなわち「リファレンス・インターフェー

スＩＤ」をあらわすパラメータ。

（３）出力パラメータの”ｐｐｖ”、すなわち、上記２つの入力パラメータを元に得られた、実体としてのインターフェースポインタ。

【００８４】

ＤＬＬ内にコードとして記述されたＩｎｔｅｒｆａｃｅ内のメソッドを、ＥＸＥが呼び出すためには、先にＤｌｌＧｅｔＣｌａｓｓＯｂｊｅｃｔ関数を用いて、クラスＩＤ及びリファレンス・インターフェースＩＤを元にしたＩｎｔｅｒｆａｃｅのインスタンスを生成し、そのインスタンスのメモリアドレスを知っておく必要がある。

【００８５】

図２７に示された”ｉｉｄ＿ｉｓ”（２７０３）が、本実施形態の特徴を最もよくあらわすものである。すなわち、ｉｉｄ＿ｉｓに第二引数であるｒｉｉｄを指定することで「この出力パラメータは、実際にはｖｏｉｄ＊ではなく、ｒｉｉｄに指定された型のインターフェースである」という定義を行なっている。この定義をＡＰＩトレーサが認識することにより、従来取得できなかった「ｐｐｖで返ってきたインターフェース内に実装されたメソッドの呼び出し」に関しても、詳細が処理ログとして取得できるようになっている。

【００８６】

図２８は、ｒｉｉｄに指定するインターフェース型を定義した、関数定義ファイルの例である。ここでは、ＩＵｎｋｎｏｗｎという名前のインターフェースから派生した、ＩＧｅｔＩｎｆｏというインターフェースを定義している。ＩＧｅｔＩｎｆｏには、ＧｅｔＮａｍｅ及びＦｒｅｅＮａｍｅＢｕｆｆｅｒという２つのメソッドが実装されている。

【００８７】

図２９は、本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

【００８８】

処理が開始されると（ステップＳ２９００）、設定された関数／メソッドが呼び出される都度に、ＤＬＬ名／インターフェース名／関数名／メソッド名／呼び

出し時の時刻をＨＤＤに保存し（ステップＳ２９０１）、その呼び出しに対してのパラメータをＨＤＤに保存する（ステップＳ２９０２）。続いて、入力パラメータの関数定義ファイルに、図２７によって説明したｉｉｄ＿ｉｓ、すなわち「リファレンス・インターフェースＩＤの定義」があるかどうかを判別し（ステップＳ２９０３）、存在する場合は、そのリファレンス・インターフェースＩＤを、メモリ上の別のエリアに保存しておく（ステップＳ２９０４）。

【００８９】

本体の関数／メソッドを呼び出し（ステップＳ２９０５）、メソッド内部の処理が終了すると、ＤＬＬ名／インターフェース名／関数名／メソッド名／終了時の時刻をＨＤＤに保存し（ステップＳ２９０６）、その呼び出しに対してのパラメータ及び戻り値をＨＤＤに保存する（ステップＳ２９０７）。

【００９０】

次に、出力パラメータがインターフェースへのポインタとして定義されているかどうかを、図２７に示した関数定義ファイルに基づいて判断し（ステップＳ２９０８）、定義されている場合は、リファレンス・インターフェースＩＤがメモリ上に保存されているかどうかを判断する（ステップＳ２９０９、ステップＳ２９１０）。リファレンス・インターフェースＩＤが保存されていない場合には、その出力パラメータをインターフェースそのものへのポインタとして認識し、メモリ上にそのポインタに対するログ取得コードを生成して（ステップＳ２９１１）、該当するパラメータ／戻り値を、ログ取得コードのメモリアドレスに書き換える（ステップＳ２９１２）。リファレンス・インターフェースＩＤが保存されている場合、その出力パラメータを、リファレンス・インターフェースＩＤがあらわすインターフェースへ派生した形のポインタ、例えば図２８に示したＩＧｅｔＩｎｆｏとして認識し、メモリ上にそのポインタに対するログ取得コードを生成して（ステップＳ２９１３）、該当するパラメータ乃至戻り値を、ログ取得コードのメモリアドレスに書き換える（ステップＳ２９１４）。

【００９１】

ステップＳ２９１２またはステップＳ２９１４の処理が終了するか、もしくはステップＳ２９０８においてインターフェースポインタとして定義されていない

ことを判断した場合は、関数／メソッドのリターン処理を行なう（ステップＳ２９１５）。

【００９２】

この処理は、評価対象となるプログラムが終了するまで（ステップＳ２９１６）続行される。

【００９３】

図３０は、図２９における処理によって取得された処理ログの例である。図２７におけるｉｉｄ＿ｉｓの定義により、ＤｌｌＧｅｔＣｌａｓｓＯｂｊｅｃｔの出力パラメータｐｐｖをＩＧｅｔＩｎｆｏ型のインターフェースとして認識した結果、ＤｌｌＧｅｔＣｌａｓｓＯｂｊｅｃｔのログだけでなく、ＩＧｅｔＩｎｆｏインターフェース内に実装されたメソッドの呼び出しに関しても、処理ログとして取得することが可能となっている。なお本実施形態は、リファレンス・インターフェースＩＤを認識することを特徴とするものであり、実際のリファレンス・インターフェースＩＤ（３００１）とＩＧｅｔＩｎｆｏとの関連付けは、公知の技術を元に行なわれる。例えば、Ｗｉｎｄｏｗｓ（登録商標）　ＯＳにおいては、この情報はレジストリに置かれており、レジストリ情報を元に関連付けが可能である。ただし、他の方法を用いて関連付けを行なっても構わない。

【００９４】

以上の説明から明らかなように、本実施形態によれば、関数定義ファイルとして、「ある入力パラメータが、出力パラメータの型をあらわす」ことを指定することにより、ソフトウェアに対する更に詳細な処理ログを取得することが可能となり、バグ原因特定など、より詳細な解析が可能となる。

【００９５】

【第４の実施形態（その２）】

上記第４の実施形態（その１）においては、実体すなわちインスタンスとしてのインターフェースの解決を行なっているが、これだけでは「インスタンスを生成する元となっている情報」すなわち、オブジェクト指向プログラミングにおける「クラス情報」に基づいた情報を、処理ログとして取得できない。そこで本実施形態では、クラス情報を追加情報として処理ログに取得するために、クラスＩ

ＤをインターフェースＩＤと関連付ける場合について説明する。

【００９６】

本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける関数定義ファイルでは、クラスＩＤは、図２７におけるＤｌｌＧｅｔＣｌａｓｓＯｂｊｅｃｔのｒｃｌｓｉｄをもって定義し、その定義のために「ｃｌｓｉｄ＿ｉｓ」という関数定義を用いる。この「ｃｌｓｉｄ＿ｉｓ」とは、「このＤｌｌＧｅｔＣｌａｓｓＯｂｊｅｃｔにおいては、ｃｌｓｉｄ＿ｉｓで指定されたパラメータをクラスＩＤとするクラス情報を元にしてインターフェースのインスタンスが生成されている」ということを指定するための関数定義である。このように関数定義ファイルを記述することにより、本実施形態にかかるソフトウェア評価システムは、より詳細な情報をログとして取得することが可能となっている。

【００９７】

図３１及び図３２は、本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートであり、図２９とは別の一例である。

【００９８】

処理が開始されると（ステップＳ３１００）、設定された関数／メソッドが呼び出される都度に、ＤＬＬ名／インターフェース名／関数名／メソッド名／呼び出し時の時刻をＨＤＤに保存し（ステップＳ３１０１）、その呼び出しに対してのパラメータをＨＤＤに保存する（ステップＳ３１０２）。続いて、入力パラメータの関数定義ファイルに、図２７及び上記によって説明したｃｌｓｉｄ＿ｉｓ、すなわち「クラスＩＤの定義」があるかどうかを判別し（ステップＳ３１０３）、存在する場合は、そのクラスＩＤをメモリ上の別のエリアに保存しておく（ステップＳ３１０４）。続いて、入力パラメータの関数定義ファイルに、図２７によって説明したｉｉｄ＿ｉｓ、すなわち「リファレンス・インターフェースＩＤの定義」があるかどうかを判別し（ステップＳ３１０５）、存在する場合は、そのリファレンス・インターフェースＩＤをメモリ上の別のエリアに保存しておく（ステップＳ３１０６）。

【００９９】

本体の関数／メソッドを呼び出し（ステップＳ３１０７）、メソッド内部の処理が終了すると、ＤＬＬ名／インターフェース名／関数名／メソッド名／終了時の時刻をＨＤＤに保存し（ステップＳ３１０８）、その呼び出しに対してのパラメータ／戻り値をＨＤＤに保存する（ステップＳ３１０９）。次に、出力パラメータがインターフェースへのポインタとして定義されているかどうかを、図２７に示した関数定義ファイルに基づいて判断し（ステップＳ３１１０）、定義されている場合は、リファレンス・インターフェースＩＤがメモリ上に保存されているかどうかを判断する（図３２のステップＳ３２００、Ｓ３２０１）。リファレンス・インターフェースＩＤが保存されていない場合、その出力パラメータを、インターフェースそのものへのポインタとして認識し、メモリ上にそのポインタに対するログ取得コードを生成して（ステップＳ３２０３）、該当するパラメータ／戻り値を、ログ取得コードのメモリアドレスに書き換える（ステップＳ３２０４）。リファレンス・インターフェースＩＤが保存されている場合、その出力パラメータをリファレンス・インターフェースＩＤがあらわすインターフェースへ派生した形のポインタ、例えば図２８に示したＩＧｅｔＩｎｆｏとして認識し、メモリ上にそのポインタに対するログ取得コードを生成して（ステップＳ３２０５）、該当するパラメータ／戻り値を、ログ取得コードのメモリアドレスに書き換える（ステップＳ３２０６）。

【０１００】

ステップＳ３２０４またはステップＳ３２０６の処理が終了すると、メモリにクラスＩＤが保存されているかを判別し（ステップＳ３２０７）、保存されている場合は、保存されたクラスＩＤをログ取得コードと関連付けてメモリに保持しておく（ステップＳ３２０８）。続いて、関数／メソッドのリターン処理を行なう（図３１のステップＳ３１１２）。この処理は、評価対象となるプログラムが終了するまで（ステップＳ３１１３）続行される。

【０１０１】

ステップＳ３２０８においてクラスＩＤを保持しておくことにより、ＩＧｅｔＩｎｆｏ内の関数が呼ばれた際に、そのクラス情報を元に、ステップＳ３１０１においてより詳細な情報をログに取得することが可能となる。図３３は、図３０

に対して、クラスＩＤを元にモジュール名が追加でログ取得可能となっていることをあらわしている（３３０３、３３０４、３３０５、３３０６）。

【０１０２】

なお本実施形態は、クラスＩＤをインターフェース・パラメータと関連付けつつ、リファレンス・インターフェースＩＤを認識することを特徴とするものであり、実際のクラスＩＤ（３０００）と、モジュール名等詳細情報との関連付けは、公知の技術を元に行なわれる。例えば、Ｗｉｎｄｏｗｓ（登録商標）　ＯＳにおいては、この情報はレジストリに置かれており、レジストリ情報を元に関連付けが可能である。が、他の方法を用いて関連付けを行なっても構わない。

【０１０３】

以上の説明から明らかなように、本実施形態によれば、関数定義ファイルにおいて「ある入力パラメータが、出力パラメータの型をあらわす」ことを指定することにより、ソフトウェアに対する更に詳細な処理ログを取得することが可能となり、バグ原因特定など、より詳細な解析が可能となる。

【０１０４】

【第５の実施形態（その１）】

上記第４の実施形態によれば、特殊なＣＯＭに対するログを取得することが可能となるが、かかる特殊なＣＯＭの場合、処理ログの対象となるモジュール名がレジストリに定義されていないため、モジュール名に基づいて処理ログを取得することができない。そこで、本実施形態では個々のインターフェース名（ＣＯＭの処理内容）に対するモジュール名を追加定義として用意し、処理ログを作成する際にインターフェース名からモジュール名を判断して書き込むこととする。

【０１０５】

図３４は、本発明の第５の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける関数定義ファイルの一例であり、一般的に広く用いられているＩＤＬにより記述されている。本ソフトウェア評価システムに於いては、このＩＤＬをトークン化したタイプライブラリ・ファイルを関数定義ファイルとして使用する。

【０１０６】

ＩＤＬはインターフェースの定義に用いられるものであり、標準ではモジュール名の定義は不可能である。例えば、複数のモジュールが、同じインターフェース定義でインターフェースを提供しており、それらが選択的にプラグイン・モジュールとして用いられている場合、インターフェース定義を行なっても、それらのインターフェースがどのプラグイン・モジュールに属するものであるかは、それらプラグイン・モジュールを選択的に呼び出しているモジュールのみが知り得る。例えば、オペレーティング・システムが選択的にドライバ用のプラグイン・モジュールを呼び出している場合、オペレーティング・システムが選択を行なうアルゴリズムが公開されていない場合は、モジュール名を関連付けて呼び出しに関する詳細情報を処理ログに取得することは不可能である。具体的には、Ｍｉｃｒｏｓｏｆｔ　Ｗｉｎｄｏｗｓ（登録商標）オペレーティング・システムに搭載されている、ユニバーサル・プリンタドライバのプラグイン・モジュールなどが考えられる。

【０１０７】

ただし、これらのプラグイン・モジュールは、ユーザの操作に関連付けられて選択的に呼び出されているケースが多い。例えば、プリンタドライバにおいては、複数搭載されたプリンタのうち、ユーザがどのプリンタで出力するのかを選択して、その選択操作に対して１対１に対応するプラグイン・モジュールが呼び出される。この場合、ＩＤＬによるインターフェース定義とは別途「今回の操作に対してはこのモジュールが呼び出される」ということを知り得て、尚且つその情報を、ログ取得プログラムに対し指示する手段があれば、その手段を用いて、上記プラグイン・モジュールのモジュール名も含めた形の処理ログを取得することが可能となる。

【０１０８】

図３５は、ライブラリ名に対してモジュール名を定義した関数定義ファイルの例であり、３５００でライブラリ名を、３５０１でモジュールの置かれているフォルダ名を、３５０２でモジュール名をそれぞれ定義している。

【０１０９】

図３６は、インターフェース名に対してモジュール名を定義した関数定義ファ

イルの例であり、３６００でインターフェース名を、３６０１でモジュールの置かれているフォルダ名を、３６０２でモジュール名をそれぞれ定義している。

【０１１０】

なお、本実施形態では、図３５乃至図３６におけるモジュール名の追加定義を、単なるテキストファイルで示しているが、本実施形態はＩＤＬのような「ＤＬＬ／インターフェース／関数／メソッドを定義するための標準フォーマット」に対して別のフォーマットの追加定義を行ない、それを元に関数／メソッドのログを取得することを特徴とするものであり、追加定義は単なるテキストファイルではなく、ＨＴＭＬやＸＭＬといったマークアップ言語で記述されていても構わない。

【０１１１】

図３７は、本発明の第５の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートであり、本実施形態の特徴を最もよくあらわすものである。

【０１１２】

処理が開始されると（ステップＳ３７００）、設定された関数／メソッドが呼び出される都度に、ＤＬＬ名／インターフェース名／関数名／メソッド名／呼び出し時の時刻をＨＤＤに保存し（ステップＳ３７０１）、その呼び出しに対してのパラメータをＨＤＤに保存する（ステップＳ３７０２）。続いて、入力パラメータの関数定義ファイルに、図３５乃至図３６によって説明した、モジュール名の追加定義が存在するかどうかを判別し（ステップＳ３７０３）、存在する場合は、そのモジュール名をメモリ上の別のエリアに保存しておく（ステップＳ３７０４）。

【０１１３】

本体の関数／メソッドを呼び出し（ステップＳ３７０５）、メソッド内部の処理が終了すると、ＤＬＬ名／インターフェース名／関数名／メソッド名／終了時の時刻をＨＤＤに保存し（ステップＳ３７０６）、その呼び出しに対してのパラメータ及び戻り値をＨＤＤに保存する（ステップＳ３７０７）。続いて、関数／メソッドのリターン処理を行なう（ステップＳ３７０８）。

【０１１４】

この処理は、評価対象となるプログラムが終了するまで（ステップＳ３７０９）続行される。

【０１１５】

このように、モジュール名を関数定義とは別のフォーマットで追加定義して、それを元に処理ログの取得を行なうことにより、例えばプラグイン・モジュールがユーザの操作に関連付けられて選択的に呼び出されているケースに対して、プラグイン・モジュールのモジュール名も含めた形の処理ログを取得することが可能となる。この機能を持つＡＰＩトレーサにより、ソフトウェアに対してモジュール名を必ず含んだ形での処理ログが取得可能となり、バグ原因特定など、より詳細な解析が可能となる。

【０１１６】

【第５の実施形態（その２）】

上記第５の実施形態（その１）においては、モジュール名の追加定義をテキストファイルで行なっているが、これに限られず、モジュール定義をユーザが選択できるようなユーザー・インターフェースを備えてもよい。実現方法としては、図３５乃至図３６におけるＰａｔｈ及びＭｏｄｕｌｅＮａｍｅの設定を、ユーザが選択可能なユーザー・インターフェースを備えても構わないし、全体のモジュール名追加定義テキストファイルを複数用意して、それらのファイルを選択するユーザ・インターフェースを備えても構わない。

【０１１７】

【第５の実施形態（その３）】

上記第５の実施形態（その１）および（その２）においては、モジュール名の追加定義を、ユーザが指定乃至選択しているが、これに限られず、追加のモジュール定義をオペレーティング・システムが備えるインターフェースから取得しても構わない。例えば、Ｍｉｃｒｏｓｏｆｔの．ＮＥＴ　Ｆｒａｍｅｗｏｒｋでは、各モジュールがＸＭＬ形式のモジュール定義データを含み、モジュール定義データをオペレーティング・システムのインターフェースで取得できる。また、この場合に、オペレーティング・システムから取得したモジュール定義に示された

モジュールに対して、インポート関数アドレステーブル及びエクスポート関数アドレステーブルなどの、モジュール内部に用意されたヘッダ情報を元に、メモリ上でそのモジュール内に実装された関数／メソッドをログ取得ルーチンにリダイレクトして、更にログ情報を詳細に取得することも可能である。例えば、ＭｉｃｒｏｓｏｆｔのＷｉｎｄｏｗｓ（登録商標）オペレーティング・システムでは、この機能によって、通常のＤＬＬに実装されたエクスポート関数の呼び出し、ＣＯＭインターフェースに実装されたメソッドの呼び出し、．ＮＥＴアセンブリに実装されたメソッドの呼び出しの３点について、１つのログデータ内に時系列ですべての情報を並べることが可能である。

【０１１８】

【他の実施形態】

なお、本発明は、複数の機器（例えばホストコンピュータ，インタフェイス機器、リーダ、プリンタなど）から構成されるシステムに適用しても、一つの機器からなる装置（例えば、複写機、ファクシミリ装置など）に適用してもよい。

【０１１９】

また、本発明の目的は、前述した実施形態の機能を実現するソフトウェアのプログラムコードを記録した記憶媒体を、システムあるいは装置に供給し、そのシステムあるいは装置のコンピュータ（またはＣＰＵやＭＰＵ）が記憶媒体に格納されたプログラムコードを読出し実行することによっても、達成されることは言うまでもない。

【０１２０】

この場合、記憶媒体から読出されたプログラムコード自体が前述した実施形態の機能を実現することになり、そのプログラムコードを記憶した記憶媒体は本発明を構成することになる。

【０１２１】

プログラムコードを供給するための記憶媒体としては、例えば、フロッピ（登録商標）ディスク、ハードディスク、光ディスク、光磁気ディスク、ＣＤ－ＲＯＭ、ＣＤ－Ｒ、磁気テープ、不揮発性のメモリカード、ＲＯＭなどを用いることができる。

【０１２２】

また、コンピュータが読出したプログラムコードを実行することにより、前述した実施形態の機能が実現されるだけでなく、そのプログラムコードの指示に基づき、コンピュータ上で稼働しているＯＳ（オペレーティングシステム）などが実際の処理の一部または全部を行い、その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。

【０１２３】

さらに、記憶媒体から読出されたプログラムコードが、コンピュータに挿入された機能拡張ボードやコンピュータに接続された機能拡張ユニットに備わるメモリに書込まれた後、そのプログラムコードの指示に基づき、その機能拡張ボードや機能拡張ユニットに備わるＣＰＵなどが実際の処理の一部または全部を行い、その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。

【０１２４】

以下に本発明の実施態様の例を示す。

【０１２５】

［実施態様１］　所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に呼び出されるオペレーティングシステム内の関数のうち、指定された関数を識別する工程と、

ロードされた前記所定の処理を行う関数のアドレスと前記指定されたオペレーティングシステム内の関数のアドレスとを、ログ取得のための関数のアドレスに書き換える工程と、を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程と

を備えることを特徴とするログ取得方法。

【０１２６】

［実施態様２］　前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出す際の所定の情報は、少なくとも、呼び出された関数の関数名、呼び出す際の時刻、呼び出す際のパラメータ、または呼び出す際のポインタ・パラメータの指すメモリ内容のいずれかを備えることを特徴とする実施態様１に記載のログ取得方法。

【０１２７】

［実施態様３］　前記実行結果を受け取った際の所定の情報は、少なくとも、受け取った際の時刻、受け取った際のパラメータ、受け取った際の戻り値、または受け取った際のポインタ・パラメータの指すメモリ内容のいずれかを備えることを特徴とする実施態様１に記載のログ取得方法。

【０１２８】

［実施態様４］　前記所定の処理を行う関数のアドレスは、該関数を提供するダイナミックリンクライブラリごとに、インポート関数アドレステーブルに記載されていることを特徴とする実施態様１に記載のログ取得方法。

【０１２９】

［実施態様５］　前記所定の処理を行う関数のアドレスのうち、前記ログ取得のための関数のアドレスに書き換えるアドレスを選択する工程を更に備え、

前記書き換える工程は、前記選択する工程により選択された関数のアドレスを書き換えることを特徴とする実施態様１に記載のログ取得方法。

【０１３０】

［実施態様６］　所定の１または複数のモジュールを定義する工程と、前記関数の呼び出しが当該定義したモジュールを経由したか否かを判別する工程とを更に備え、当該定義されたモジュールを経由した場合には、当該関数を呼び出す際の前記所定の情報を記録対象から除外することを特徴とする実施態様１のログ取得方法。

【０１３１】

［実施態様７］　オペレーティング・システムに含まれる所定の１または複数のモジュールを定義する工程と、前記関数の呼び出しが経由したモジュールを判別する工程とを更に備え、当該定義されたモジュール以外のオペレーティング・システムに含まれるモジュールを経由した場合には、当該関数を呼び出す際の前記所定の情報を記録対象から除外することを特徴とする実施態様１のログ取得方法。

【０１３２】

［実施態様８］　前記所定の情報が構造体である場合に、当該構造体が置かれたメモリ領域の先頭から当該構造体のサイズ分だけ後のメモリ領域に格納されている情報を、当該構造体に指定されたサイズに応じて記録することを特徴とする実施態様１のログ取得方法。

【０１３３】

［実施態様９］　前記所定の情報が構造体である場合に、当該構造体が置かれたメモリ領域の先頭から当該構造体に対して指定されたオフセットより後のメモリ領域に格納されている情報を、定義されたデータ型のデータとして記録することを特徴とする実施態様１のログ取得方法。

【０１３４】

［実施態様１０］　何れのモジュールの何れの関数の所定の情報の何れがオブジェクトの情報を表わし、何れがオブジェクトのクラスＩＤを表わすかを定義しておき、関数の呼び出しにおいて、オブジェクトのクラスＩＤに基づいて、その関数の所定の情報に格納されたオブジェクトの情報を記録することを特徴とする実施態様１のログ取得方法。

【０１３５】

［実施態様１１］　何れのモジュールの何れの関数の所定の情報の何れがオブジェクトの情報を表わし、何れがオブジェクトのインターフェースＩＤを表わすかを定義しておき、関数の呼び出しにおいて、オブジェクトのインターフェースＩＤに基づいて、その関数の所定の情報に格納されたオブジェクトの情報を記録することを特徴とする実施態様１のログ取得方法。

【０１３６】

［実施態様１２］　ライブラリまたは各インタフェースに対するモジュール名の追加定義を用意しておき、関数を呼び出す際に、当該追加定義を参照してモジュール名を記録することを特徴とする実施態様１のログ取得方法。

【０１３７】

［実施態様１３］　所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

前記所定の処理を行う関数がロードされた場合のアドレスから所定のオフセットを有する領域であって、指定された領域を識別する工程と、

前記ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程と、を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行う関数を呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程と、

前記指定された領域のデータを読み出して記録する工程と

を備えることを特徴とするログ取得方法。

【０１３８】

［実施態様１４］　所定の処理を行う第１の関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に該プログラムの実行ファイルから直接呼び出された前記第１の関数によってのみ呼び出される第２の関数のうち、指定された第２の関数を識別する工程と、

前記第１の関数によって呼び出されることでロードされる前記指定された第２の関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程と、を備え、

前記ログ取得のための関数は、

前記指定された第２の関数を呼び出し、該所定の処理を実行させ、実行結果を受け取る工程と、

前記指定された第２の関数を呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程と

を備えることを特徴とするログ取得方法。

【０１３９】

［実施態様１５］　所定の処理を行う第１の関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に、該プログラムの実行ファイルから直接呼び出された前記第１の関数によってのみ呼び出される第２の関数に設定されたＩＤを識別する工程と、

前記第１の関数によって呼び出されることでロードされる前記ＩＤが設定された第２の関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程と、を備え、

前記ログ取得のための関数は、

前記ＩＤが設定された第２の関数を呼び出し、該所定の処理を実行させ、実行結果を受け取る工程と、

前記ＩＤが設定された第２の関数を呼び出す際の所定の情報を、前記設定されたＩＤとともに記録する工程と、

前記実行結果を受け取った際の所定の情報を、前記設定されたＩＤとともに記録する工程と

を備えることを特徴とするログ取得方法。

【０１４０】

［実施態様１６］　所定の処理を行うメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に呼び出されるオペレーティングシステム内のメソッドのうち、指定されたメソッドを識別する工程と、

ロードされた前記所定の処理を行うメソッドのアドレスと前記指定されたオペレーティングシステム内のメソッドのアドレスとを、ログ取得のためのメソッドのアドレスに書き換える工程と、を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドおよび前記指定されたオペレーティングシステム内のメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行うメソッドおよび前記指定されたオペレーティングシステム内のメソッドを呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程と

を備えることを特徴とするログ取得方法。

【０１４１】

［実施態様１７］　前記所定の処理を行うメソッドおよび前記指定されたオペレーティングシステム内のメソッドを呼び出す際の所定の情報は、少なくとも、呼び出されたメソッドのメソッド名、呼び出す際の時刻、呼び出す際のパラメータ、または呼び出す際のポインタ・パラメータの指すメモリ内容のいずれかを備えることを特徴とする実施態様１６に記載のログ取得方法。

【０１４２】

［実施態様１８］　前記実行結果を受け取った際の所定の情報は、少なくとも、受け取った際の時刻、受け取った際のパラメータ、受け取った際の戻り値、または受け取った際のポインタ・パラメータの指すメモリ内容のいずれかを備えることを特徴とする実施態様１６に記載のログ取得方法。

【０１４３】

［実施態様１９］　前記所定の処理を行うメソッドのアドレスは、該メソッドを提供するインターフェースごとに、仮想アドレステーブルに記載されていることを特徴とする実施態様１６に記載のログ取得方法。

【０１４４】

［実施態様２０］　前記所定の処理を行うメソッドのアドレスのうち、前記ログ取得のためのメソッドのアドレスに書き換えるアドレスを選択する工程を更に備え、

前記書き換える工程は、前記選択する工程により選択されたメソッドのアドレスを書き換えることを特徴とする実施態様１６に記載のログ取得方法。

【０１４５】

［実施態様２１］　所定の処理を行うメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

前記所定の処理を行うメソッドがロードされた場合のアドレスから所定のオフセットを有する領域であって、指定された領域を識別する工程と、

前記ロードされた前記所定の処理を行うメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程と、を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行うメソッドを呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程と、

前記指定された領域のデータを読み出して記録する工程と

を備えることを特徴とするログ取得方法。

【０１４６】

［実施態様２２］　所定の処理を行う第１のメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に該プログラムの実行ファイルから直接呼び出された前記第１のメソッドによってのみ呼び出される第２のメソッドのうち、指定された第２のメソッドを識別する工程と、

前記第１のメソッドによって呼び出されることでロードされる前記指定された第２のメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程と、を備え、

前記ログ取得のためのメソッドは、

前記指定された第２のメソッドを呼び出し、該所定の処理を実行させ、実行結果を受け取る工程と、

前記指定された第２のメソッドを呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程と

を備えることを特徴とするログ取得方法。

【０１４７】

［実施態様２３］　所定の処理を行う第１のメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に、該プログラムの実行ファイルから直接呼び出された前記第１のメソッドによってのみ呼び出される第２のメソッドに設定されたＩＤを識別する工程と、

前記第１のメソッドによって呼び出されることでロードされる前記ＩＤが設定された第２のメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程と、を備え、

前記ログ取得のためのメソッドは、

前記ＩＤが設定された第２のメソッドを呼び出し、該所定の処理を実行させ、実行結果を受け取る工程と、

前記ＩＤが設定された第２のメソッドを呼び出す際の所定の情報を、前記設定されたＩＤとともに記録する工程と、

前記実行結果を受け取った際の所定の情報を、前記設定されたＩＤとともに記録する工程と

を備えることを特徴とするログ取得方法。

【０１４８】

［実施態様２４］　実施態様１乃至２３のいずれか１つに記載のログ取得方法をコンピュータによって実現させるための制御プログラム。

【０１４９】

［実施態様２５］　実施態様２４に記載の制御プログラムを格納した記憶媒体。

【０１５０】

【発明の効果】

以上説明したように本発明によれば、複数にモジュール分けされたソフトウェアの処理ログを容易に取得でき、かつ、ソフトウェアの障害の原因の解析のための工数を削減することが可能となる。

【図面の簡単な説明】

【図１】

本発明の第１の実施形態にかかるログ取得方法を実現するコンピュータ（ソフトウェア評価システム）の構成をあらわす図である。

【図２】

本発明の第１の実施形態にかかるログ取得方法を説明するためにあらわした、関数ロード時の通常のメモリ構成をあらわす図である。

【図３】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのＩＡＴ　Ｐａｔｃｈ使用時のメモリ構成をあらわす図である。

【図４Ａ】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのＩＡＴ　Ｐａｔｃｈを使用時の状態を示す図である。

【図４Ｂ】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのログ取得処理の流れを示す図である。

【図５】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのＩＡＴ　Ｐａｔｃｈを使用時の内部構成をあらわす図である。

【図６】

本発明の第１の実施形態にかかるログ取得方法を説明するためにあらわした、ＣＯＭサーバのインターフェースのインスタンス作成時の通常のメモリ構成をあらわす図である。

【図７】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのＶＴａｂｌｅ　Ｐａｔｃｈ使用時のメモリ構成をあらわす図である。

【図８Ａ】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのＶＴａｂｌｅ　Ｐａｔｃｈを使用時の状態を示す図である。

【図８Ｂ】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価シ

ステムのログ取得処理の流れを示す図である。

【図９】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの、内部構成をあらわす図である。

【図１０】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムに対して、それぞれの関数及びメソッドのパラメータの形式や、戻り値の形式を指示する、関数定義ファイルの例を示す図である。

【図１１】

本発明の第１の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで取得した、ログの一例を示す図である。

【図１２】

一般的なソフトウェアのモジュール構成の一例を示す図である。

【図１３】

本発明の第２の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、処理ログの収集対象となるオペレーティング・システムのモジュールを定義する例を示す図である。

【図１４】

本発明の第２の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

【図１５Ａ】

本発明の第２の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで取得した処理ログの一例を示す図である。

【図１５Ｂ】

ＯＳ部分のＤＬＬ内の関数が呼び出された際にも、処理ログを取得することとした場合の処理ログの例を示す図である。

【図１６】

本発明の第２の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、処理ログの収集対象からオペレーティング・システムの除外モ

ジュールを定義する例を示す図である。

【図１７】

本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

【図１８】

本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義の一例を示す図であり、一般的に広く用いられているＩＤＬにより、記述された図である。

【図１９】

本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義ファイルを示す図であり、構造体へのポインタに対して付加バイナリ情報取得を指定することで、ポインタ・データの実体をログとして取得すべく、ＩＤＬにより記述したファイルを示す図である。

【図２０】

図１９に示された関数定義による構造体が、どのようにメモリ上に配置されるかを表す図である。

【図２１】

本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１９に示すように関数が定義されている時のログを取得する場合の処理の流れを示すフローチャートである。

【図２２】

本発明の第３の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１９の定義により取得されたログデータを示す図である。

【図２３】

本発明の一実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義ファイルを示す図であり、構造体へのポインタに対してメンバをオフセットとして指定することで、ポインタ・データの実体をログとして取得すべく、ＩＤＬにより記述したファイルを示す図である。

【図２４】

図１８に示された関数定義による構造体が、どのようにメモリ上に配置されるかを表す図である。

【図２５】

本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１８に示すように関数が定義されている時のログを取得する場合の処理の流れを示すフローチャートである。

【図２６】

本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図１８の定義により取得されたログデータを示す図である。

【図２７】

本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムに対して、それぞれの関数及びメソッドのパラメータの形式や、戻り値の形式を指示する関数定義ファイルの例を示す図である。

【図２８】

本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、ｒｉｉｄに指定するインターフェース型を定義した、関数定義ファイルの例を示す図である。

【図２９】

本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

【図３０】

本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図２９における処理によって取得された処理ログの例を示す図である。

【図３１】

本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

【図３２】

本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価シ

ステムにおける処理の流れを示すフローチャートである。

【図３３】

本発明の第４の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図３１および図３２における処理によって取得された処理ログの例を示す図である。

【図３４】

本発明の第５の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義の一例を示す図である。

【図３５】

本発明の第５の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、ライブラリ名に対してモジュール名を定義する例を示す図である。

【図３６】

本発明の第５の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、インターフェース名に対してモジュール名を定義する例を示す図である。

【図３７】

本発明の第５の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

【符号の説明】

１：ＣＰＵ

２：チップセット

３：ＲＡＭ

４：ハードディスクコントローラ

５：ディスプレイコントローラ

６：ハードディスクドライブ

７：ＣＤ－ＲＯＭドライブ

８：ディスプレイ

１１：ＣＰＵとチップセットを繋ぐ信号線

１２：チップセットとＲＡＭを繋ぐ信号線

１３：チップセットと各種周辺機器とを繋ぐ周辺機器バス

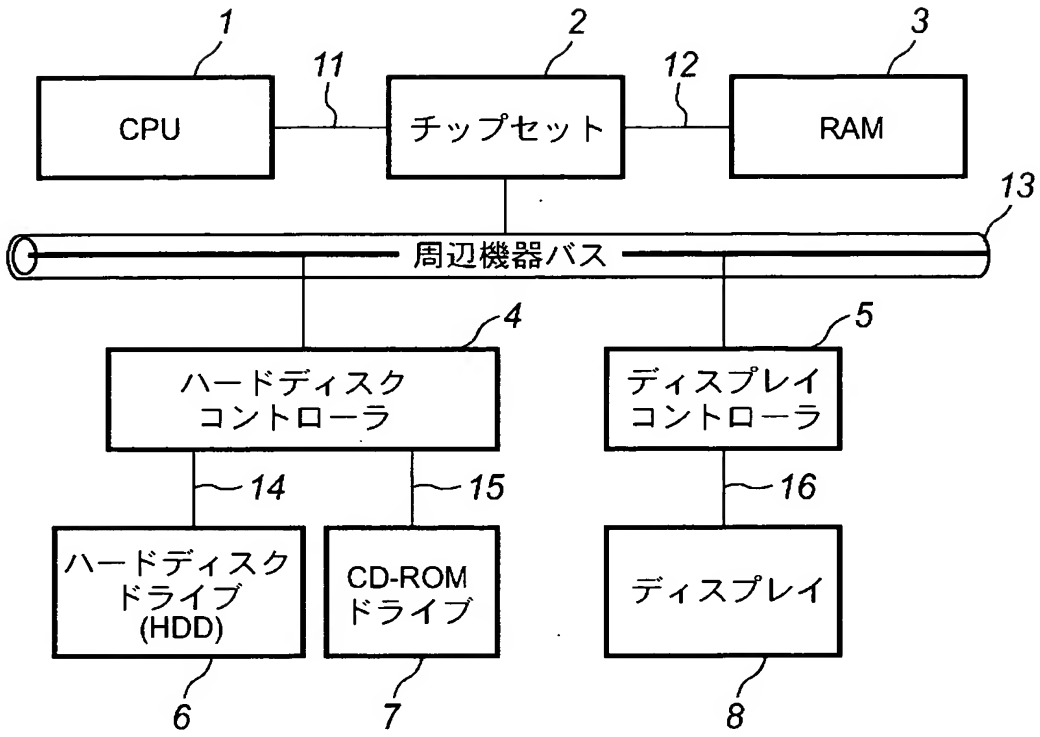
１４：ハードディスクコントローラとハードディスクドライブを繋ぐ信号線

１５：ハードディスクコントローラとＣＤ－ＲＯＭドライブを繋ぐ信号線

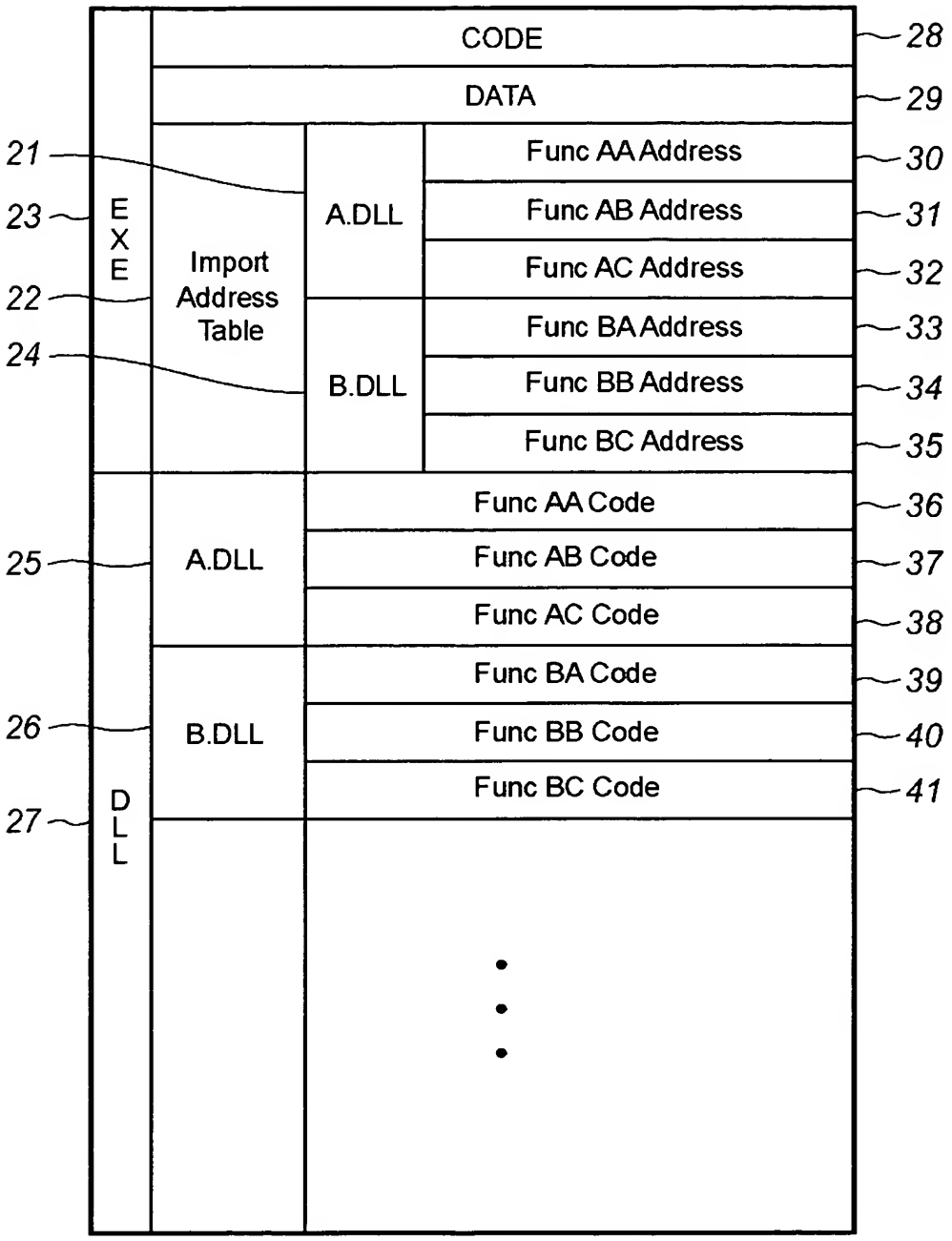
１６：ディスプレイコントローラとディスプレイを繋ぐ信号線

【書類名】　図面

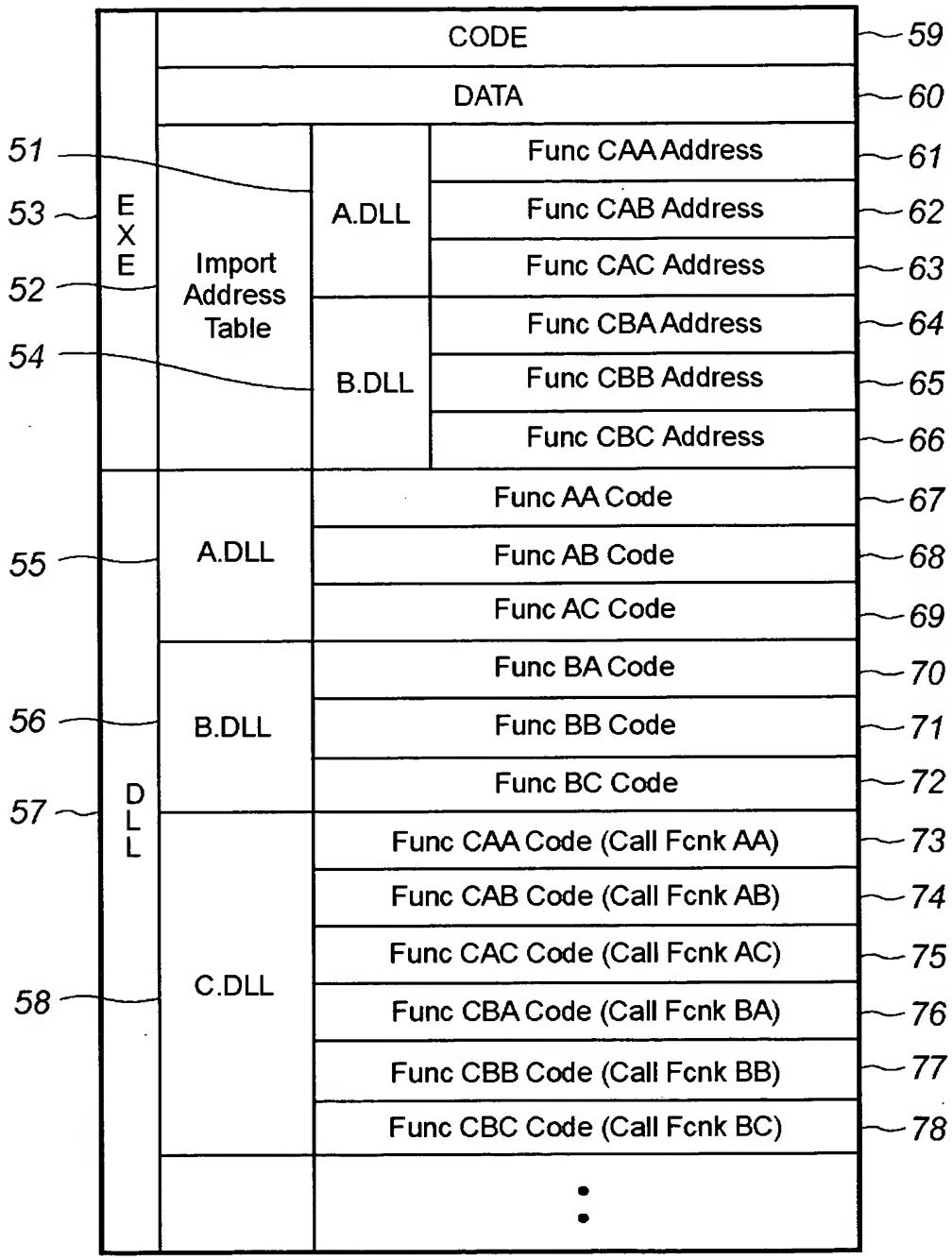
【図１】


1
2
3
11
12
CPU
チップセット
RAM
13
周辺機器バス
4
5
ハードディスク
コントローラ
ディスプレイ
コントローラ
14
15
16
ハードディスク
ドライブ
(HDD)
CD-ROM
ドライブ
ディスプレイ
6
7
8

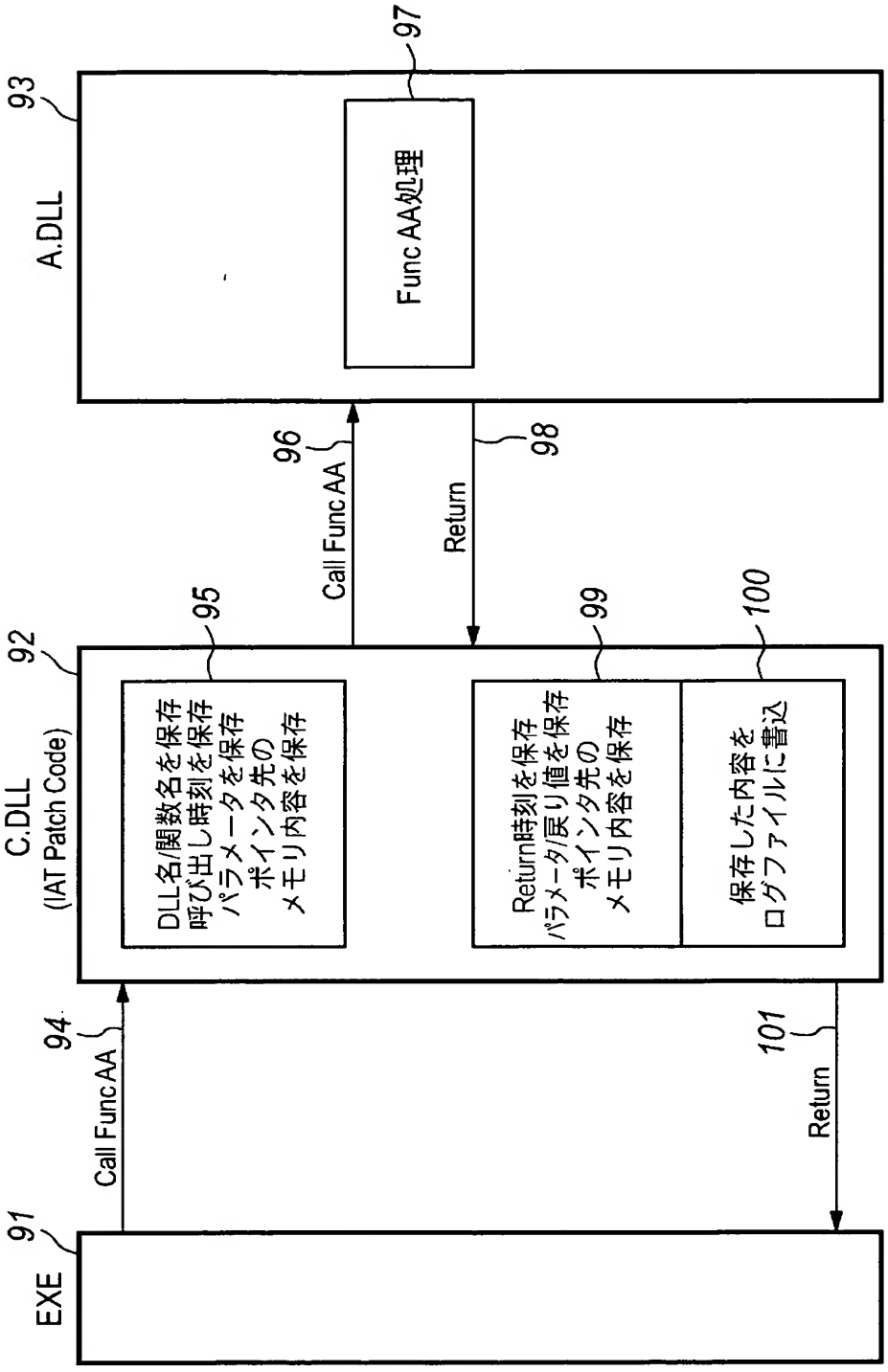
【図２】


CODE
DATA
Func AA Address
Func AB Address
Func AC Address
Func BA Address
Func BB Address
Func BC Address
Func AA Code
Func AB Code
Func AC Code
Func BA Code
Func BB Code
Func BC Code
EXE
DLL
Import Address Table
A.DLL
B.DLL
A.DLL
B.DLL
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41

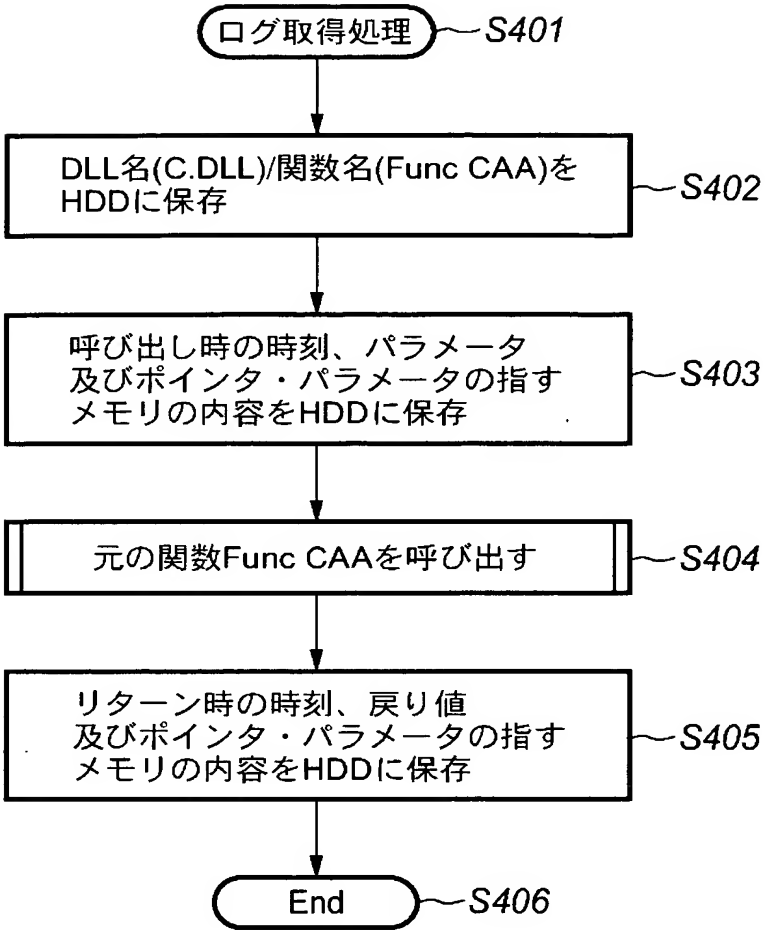
【図３】


CODE
DATA
EXE
Import Address Table
A.DLL
B.DLL
Func CAA Address
Func CAB Address
Func CAC Address
Func CBA Address
Func CBB Address
Func CBC Address
DLL
A.DLL
B.DLL
C.DLL
Func AA Code
Func AB Code
Func AC Code
Func BA Code
Func BB Code
Func BC Code
Func CAA Code (Call Fcnk AA)
Func CAB Code (Call Fcnk AB)
Func CAC Code (Call Fcnk AC)
Func CBA Code (Call Fcnk BA)
Func CBB Code (Call Fcnk BB)
Func CBC Code (Call Fcnk BC)
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78

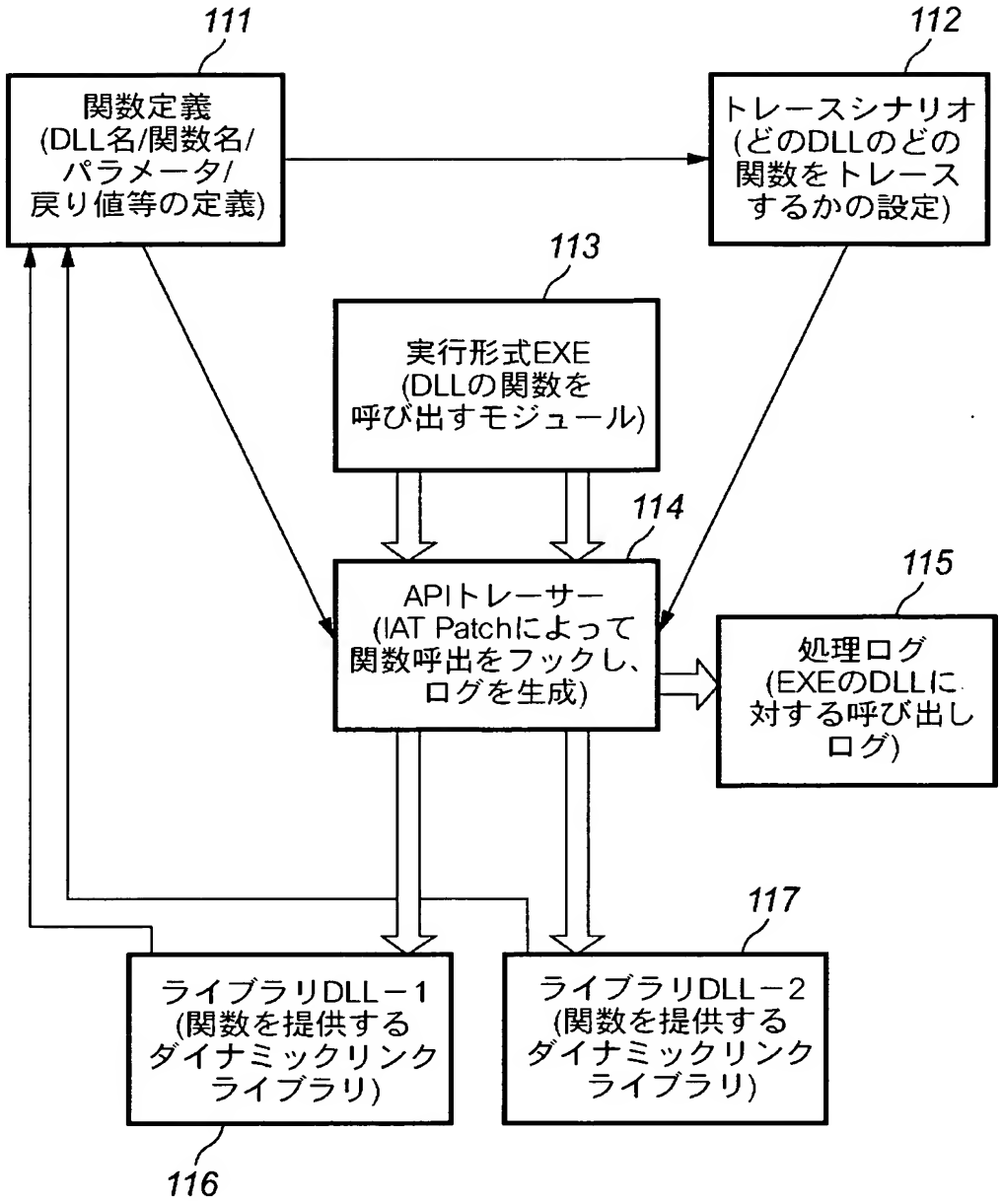
【図４Ａ】


97
93
A.DLL
Func AA処理
96
Call Func AA
98
Return
95
99
100
92
C.DLL
(IAT Patch Code)
DLL名/関数名を保存
呼び出し時刻を保存
パラメータを保存
ポインタ先の
メモリ内容を保存
Return時刻を保存
パラメータ/戻り値を保存
ポインタ先の
メモリ内容を保存
保存した内容を
ログファイルに書込
94
Call Func AA
101
Return
91
EXE

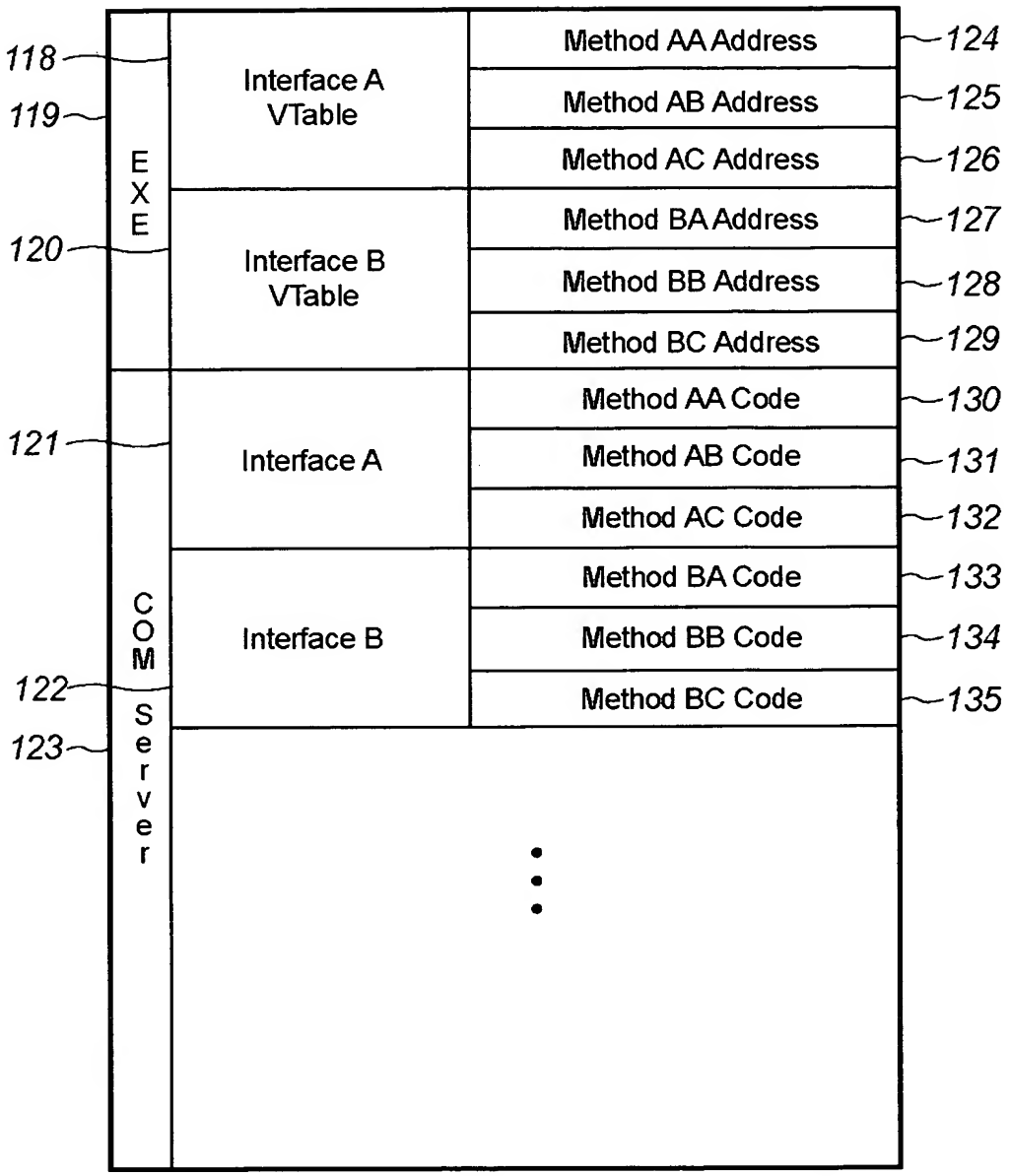
【図４Ｂ】


ログ取得処理
S401
DLL名(C.DLL)/関数名(Func CAA)を
HDDに保存
S402
呼び出し時の時刻、パラメータ
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S403
元の関数Func CAAを呼び出す
S404
リターン時の時刻、戻り値
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S405
End
S406

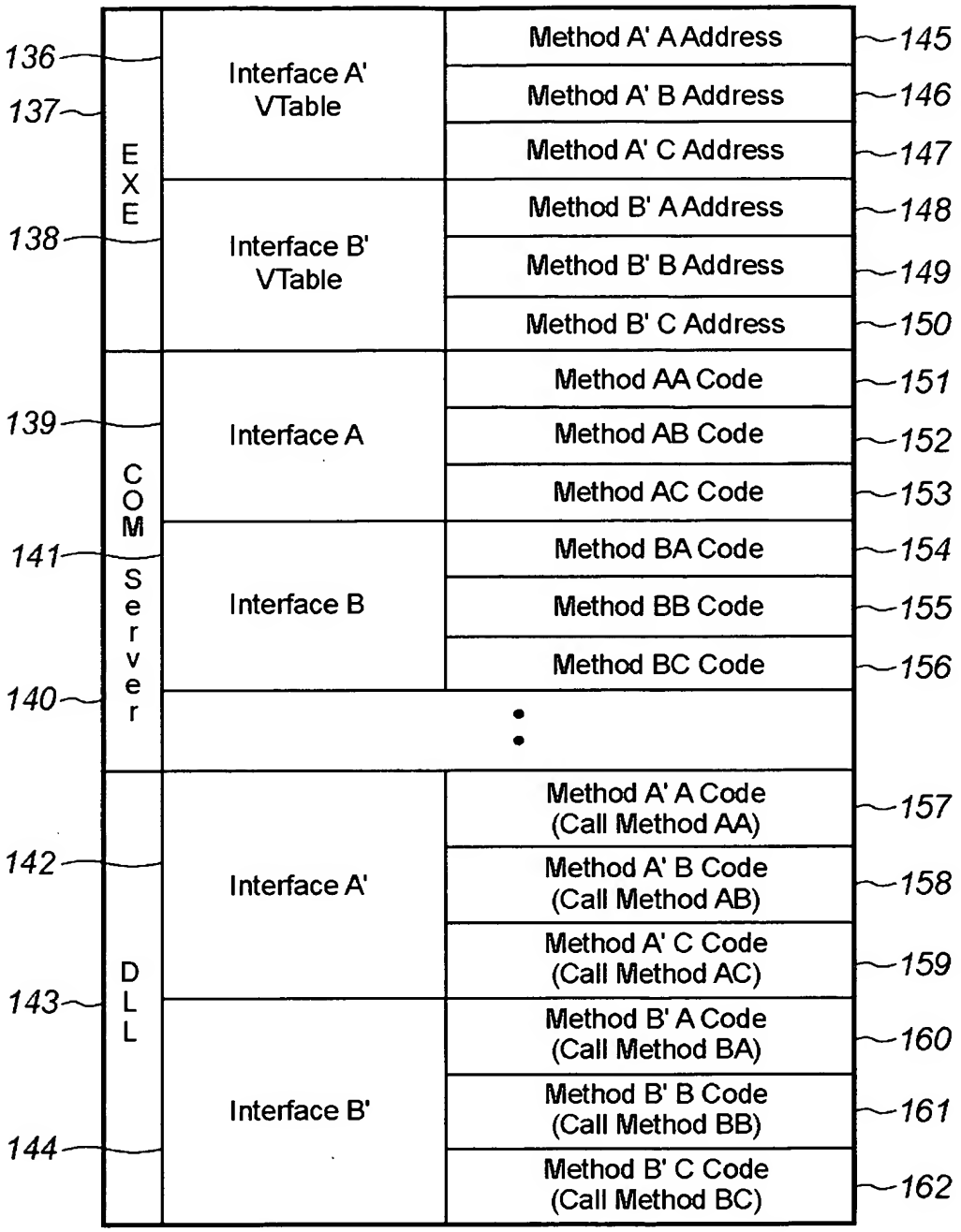
【図５】


111
関数定義
(DLL名/関数名/
パラメータ/
戻り値等の定義)
112
トレースシナリオ
(どのDLLのどの
関数をトレース
するかの設定)
113
実行形式EXE
(DLLの関数を
呼び出すモジュール)
114
APIトレーサー
(IAT Patchによって
関数呼出をフックし、
ログを生成)
115
処理ログ
(EXEのDLLに
対する呼び出し
ログ)
117
ライブラリDLL－1
(関数を提供する
ダイナミックリンク
ライブラリ)
ライブラリDLL－2
(関数を提供する
ダイナミックリンク
ライブラリ)
116

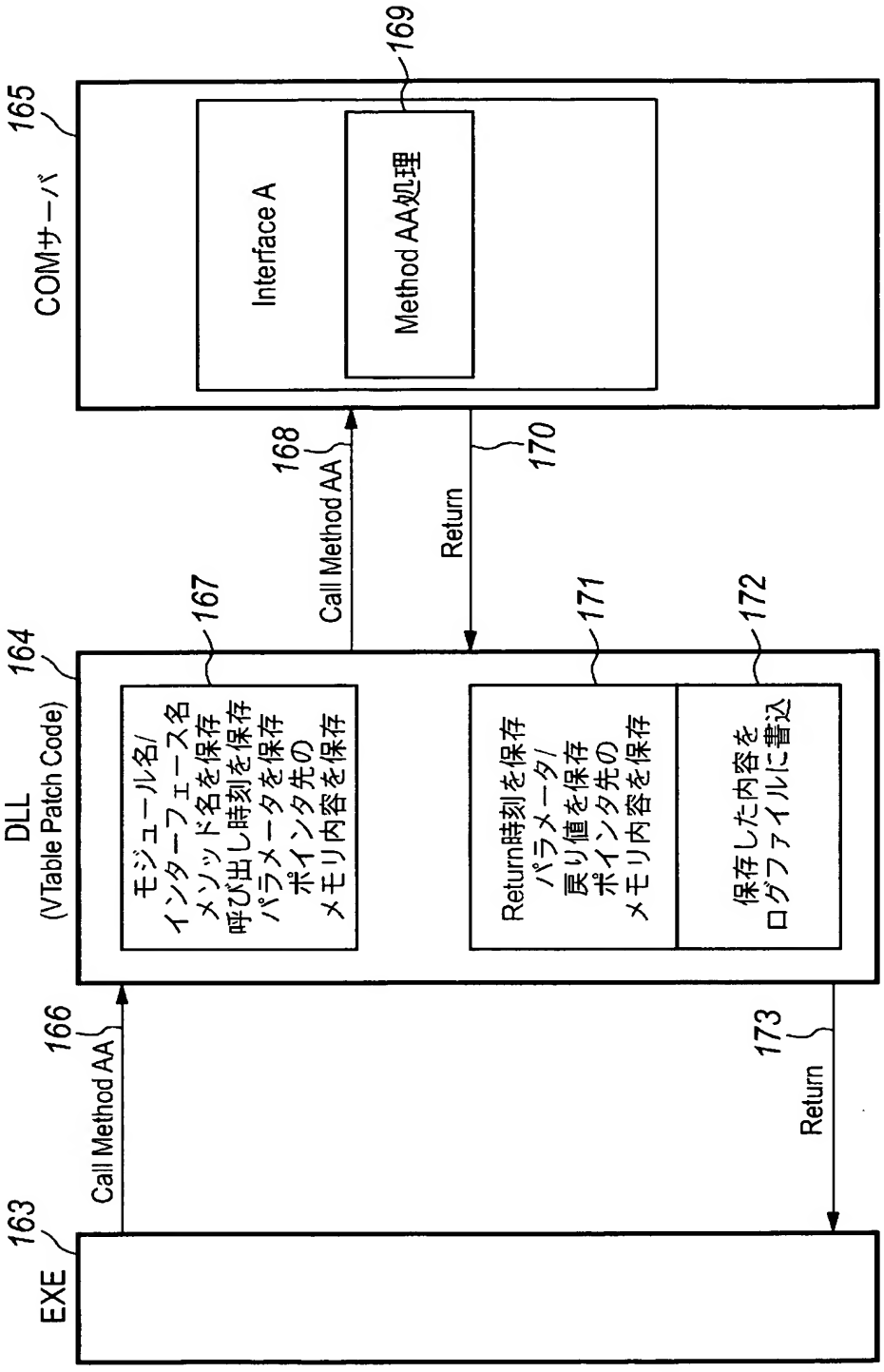
【図６】


118
119
120
121
122
123
EXE
COM Server
Interface A VTable
Interface B VTable
Interface A
Interface B
Method AA Address
Method AB Address
Method AC Address
Method BA Address
Method BB Address
Method BC Address
Method AA Code
Method AB Code
Method AC Code
Method BA Code
Method BB Code
Method BC Code
124
125
126
127
128
129
130
131
132
133
134
135

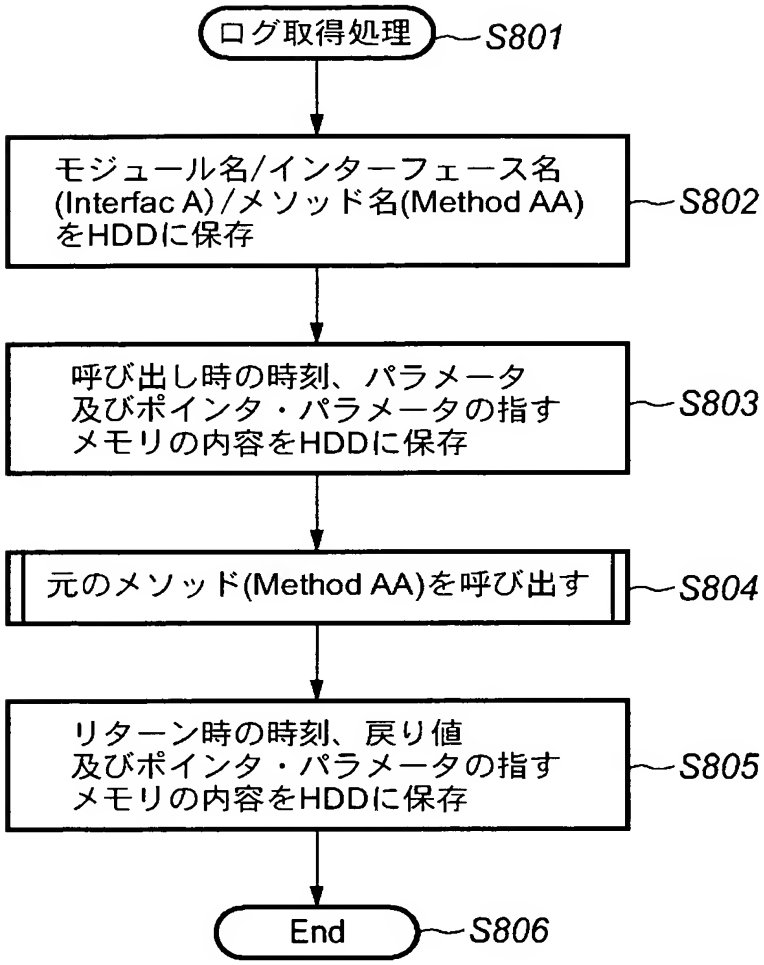
【図７】


136
137
138
139
141
140
142
143
144
EXE
COM Server
DLL
Interface A' VTable
Interface B' VTable
Interface A
Interface B
Interface A'
Interface B'
Method A' A Address
Method A' B Address
Method A' C Address
Method B' A Address
Method B' B Address
Method B' C Address
Method AA Code
Method AB Code
Method AC Code
Method BA Code
Method BB Code
Method BC Code
Method A' A Code (Call Method AA)
Method A' B Code (Call Method AB)
Method A' C Code (Call Method AC)
Method B' A Code (Call Method BA)
Method B' B Code (Call Method BB)
Method B' C Code (Call Method BC)
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162

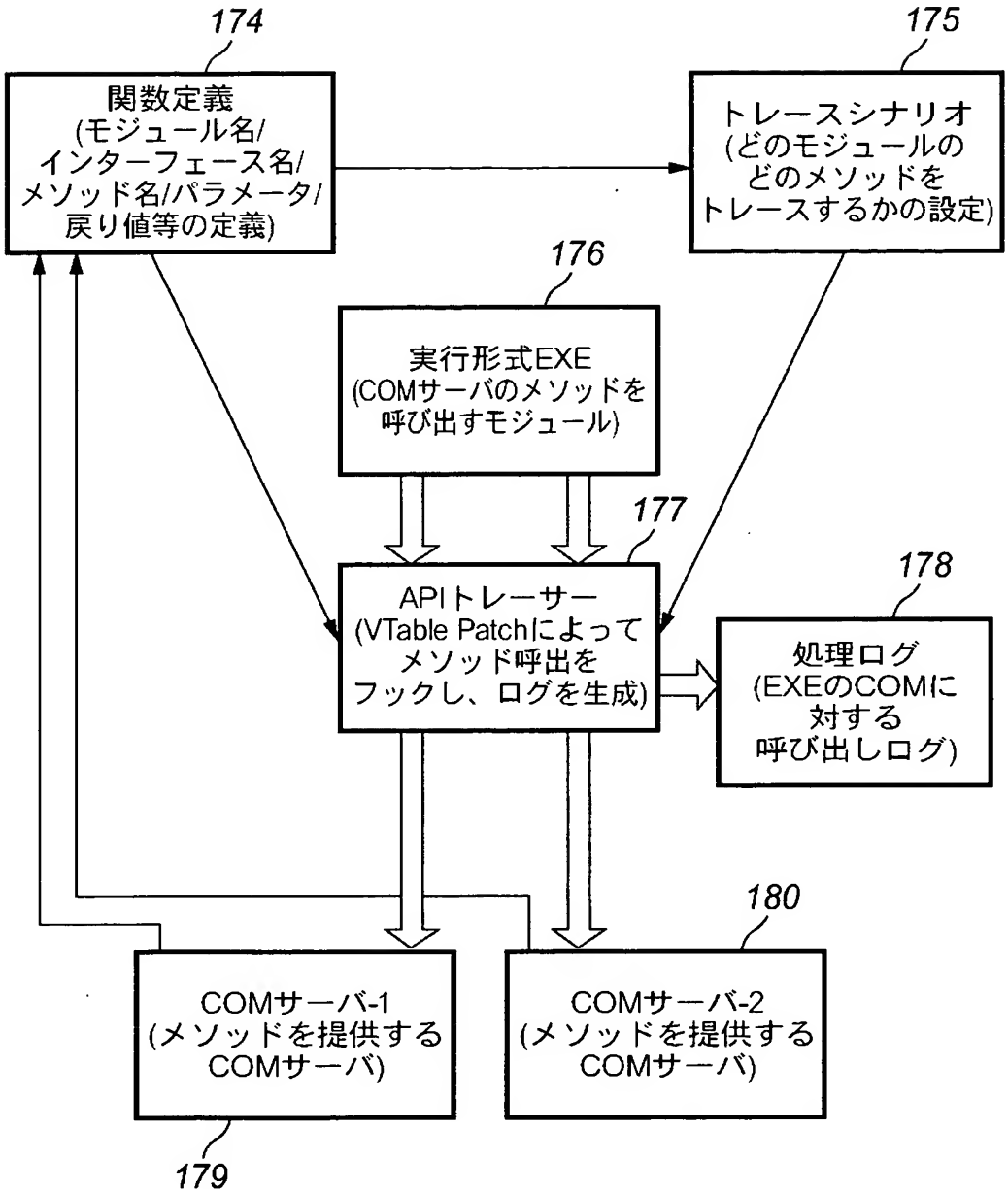
【図８Ａ】


165
COMサーバ
169
Interface A
Method AA処理
168
Call Method AA
170
Return
164
DLL
(VTable Patch Code)
167
モジュール名/
インターフェース名
メソッド名を保存
呼び出し時刻を保存
パラメータを保存
ポインタ先の
メモリ内容を保存
171
Return時刻を保存
パラメータ/
戻り値を保存
ポインタ先の
メモリ内容を保存
172
保存した内容を
ログファイルに書込
166
Call Method AA
173
Return
163
EXE

【図８Ｂ】


ログ取得処理
S801
モジュール名/インターフェース名
(Interfac A)/メソッド名(Method AA)
をHDDに保存
S802
呼び出し時の時刻、パラメータ
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S803
元のメソッド(Method AA)を呼び出す
S804
リターン時の時刻、戻り値
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S805
End
S806

【図９】


174
関数定義
(モジュール名/
インターフェース名/
メソッド名/パラメータ/
戻り値等の定義)
175
トレースシナリオ
(どのモジュールの
どのメソッドを
トレースするかの設定)
176
実行形式EXE
(COMサーバのメソッドを
呼び出すモジュール)
177
APIトレーサー
(VTable Patchによって
メソッド呼出を
フックし、ログを生成)
178
処理ログ
(EXEのCOMに
対する
呼び出しログ)
180
COMサーバ-1
(メソッドを提供する
COMサーバ)
COMサーバ-2
(メソッドを提供する
COMサーバ)
179

【図１０】

Interface名：　Interface A
Method名：Method AA
引数：　DWORD dwID
戻り値：　DWORD dwRet

Interface名：　Interface A
Method名：Method AB
引数：　DWORD dwSize
戻り値：　int　nRet

Interface名：　Interface A
Method名：Method AC
引数：　DWORD dwSize
戻り値：　DWORD dwRet

Interface名：　Interface B
Method名：Method BA
引数：　DWORD dwSize
戻り値：　int　nRet

Interface名：　Interface B
Method名：Method BC
引数：　DWORD dwID
戻り値：　DWORD dwRet

【図１１】

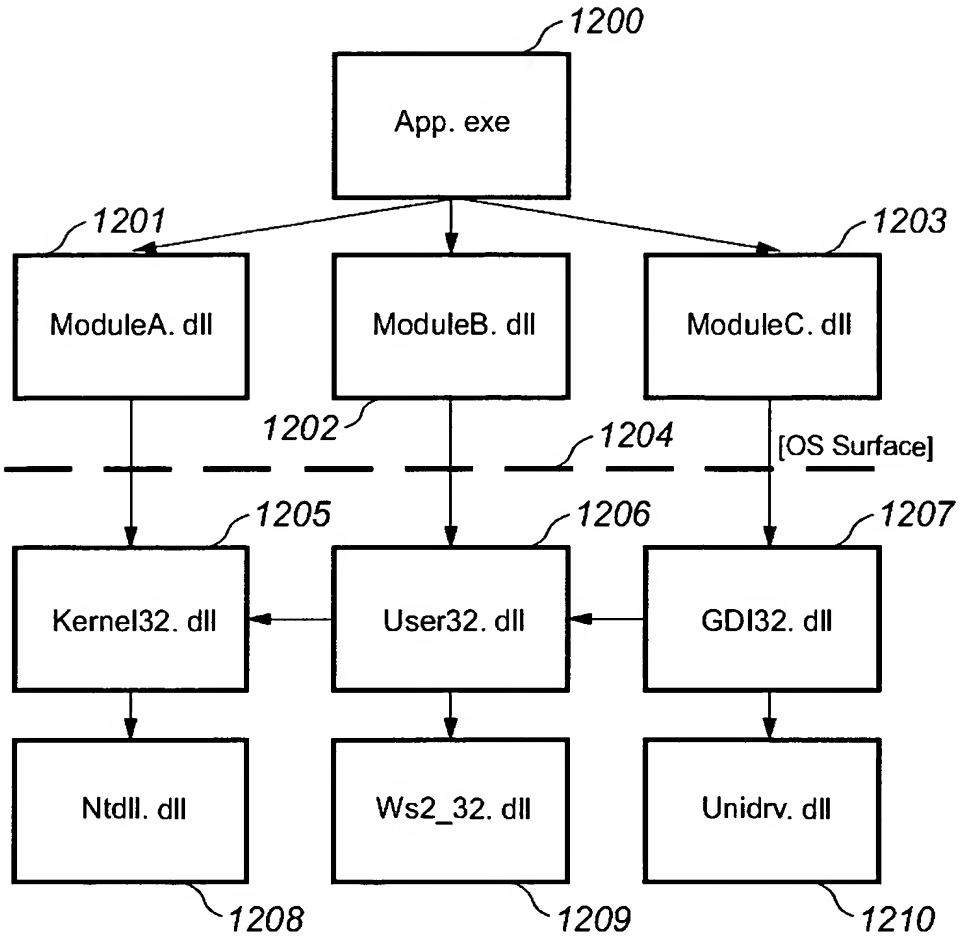
Interface名：　Interface A
Method名：Method AA
引数：　DWORD dwID：256
戻り値：　DWORD dwRet：0
In時刻：　2002/03/25 22：24：12.025
Out時刻：　2002/03/25 22：24：12.035

Interface名：　Interface B
Method名：Method BA
引数：　DWORD dwSize：512
戻り値：　int nRet：-1
In時刻：　2002/03/25 22：24：12.046
Out時刻：　2002/03/25 22：24：12.057

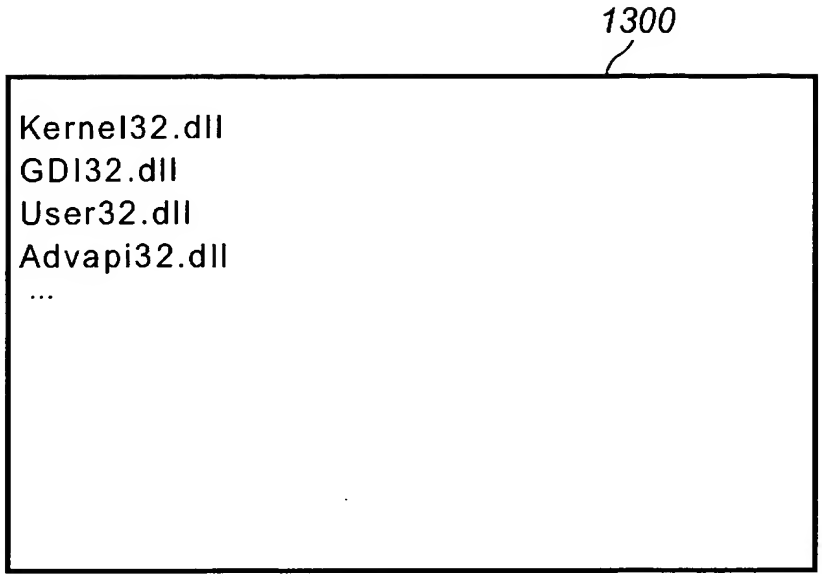
Interface名：　Interface B
Method名：Method BC
引数：　DWORD dwID：12
戻り値：　DWORD dwRet：2
In時刻：　2002/03/25 22：24：12.068
Out時刻：　2002/03/25 22：24：12.079

...

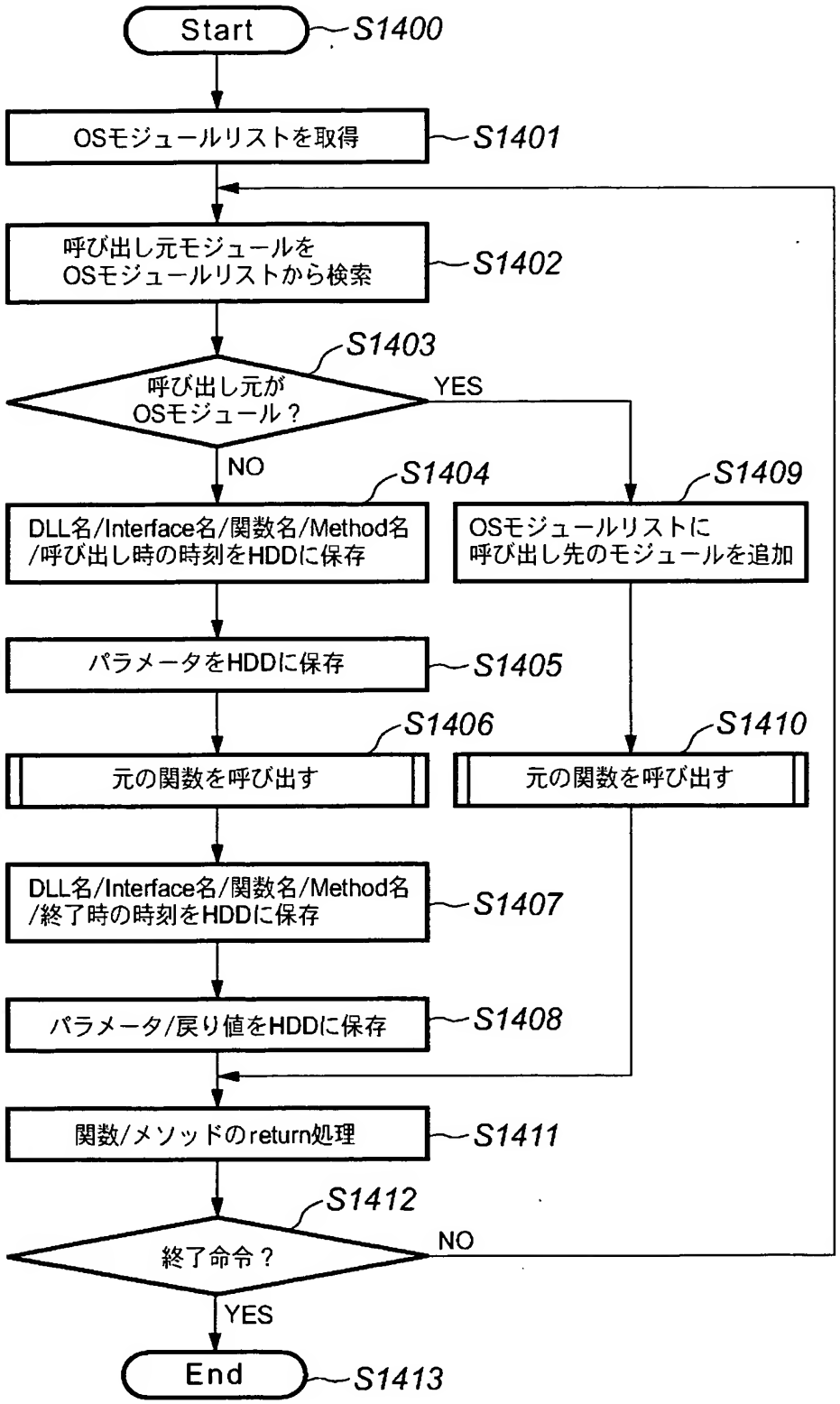
【図１２】


1200
App. exe
1201
1203
ModuleA. dll
ModuleB. dll
ModuleC. dll
1202
1204
[OS Surface]
1205
1206
1207
Kernel32. dll
User32. dll
GDI32. dll
Ntdll. dll
Ws2_32. dll
Unidrv. dll
1208
1209
1210

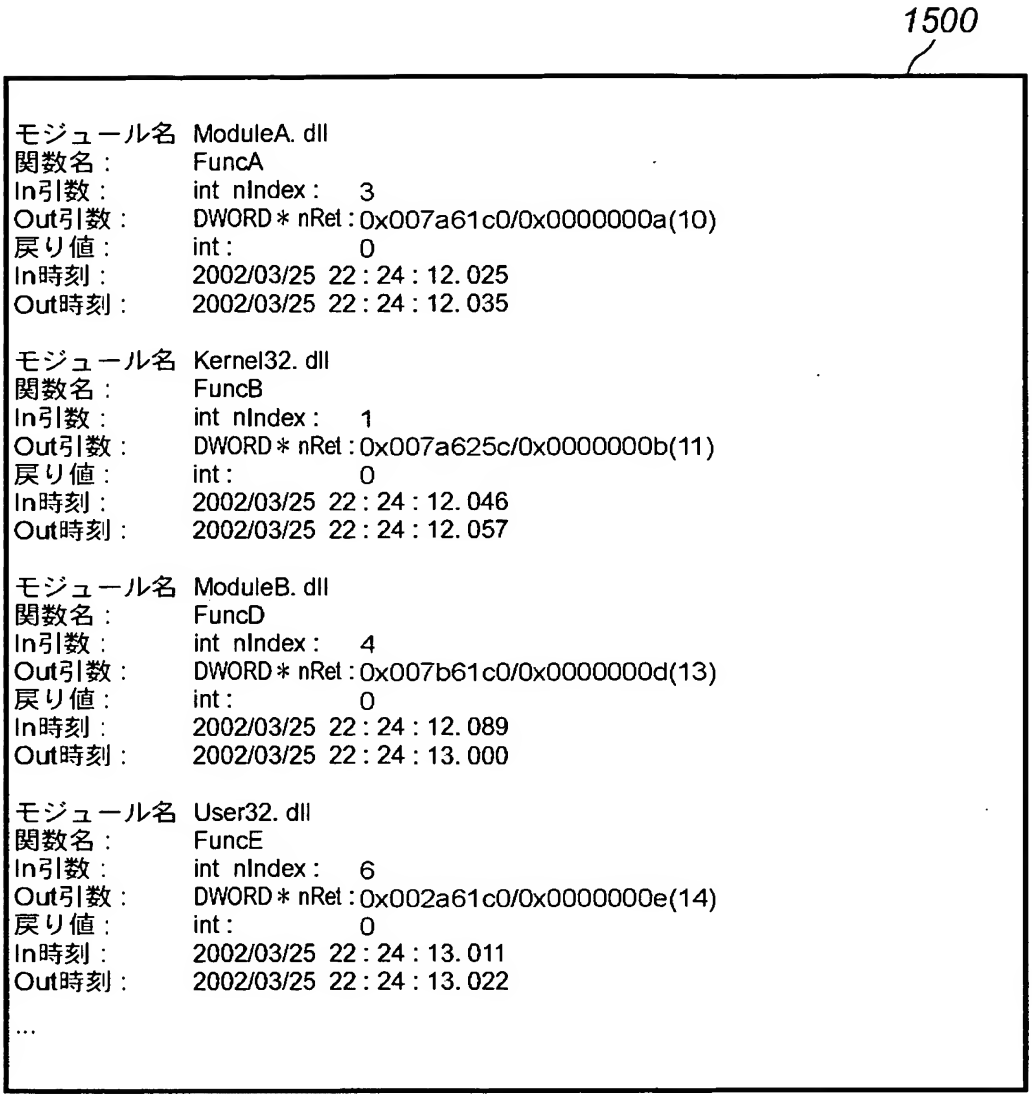
【図１３】



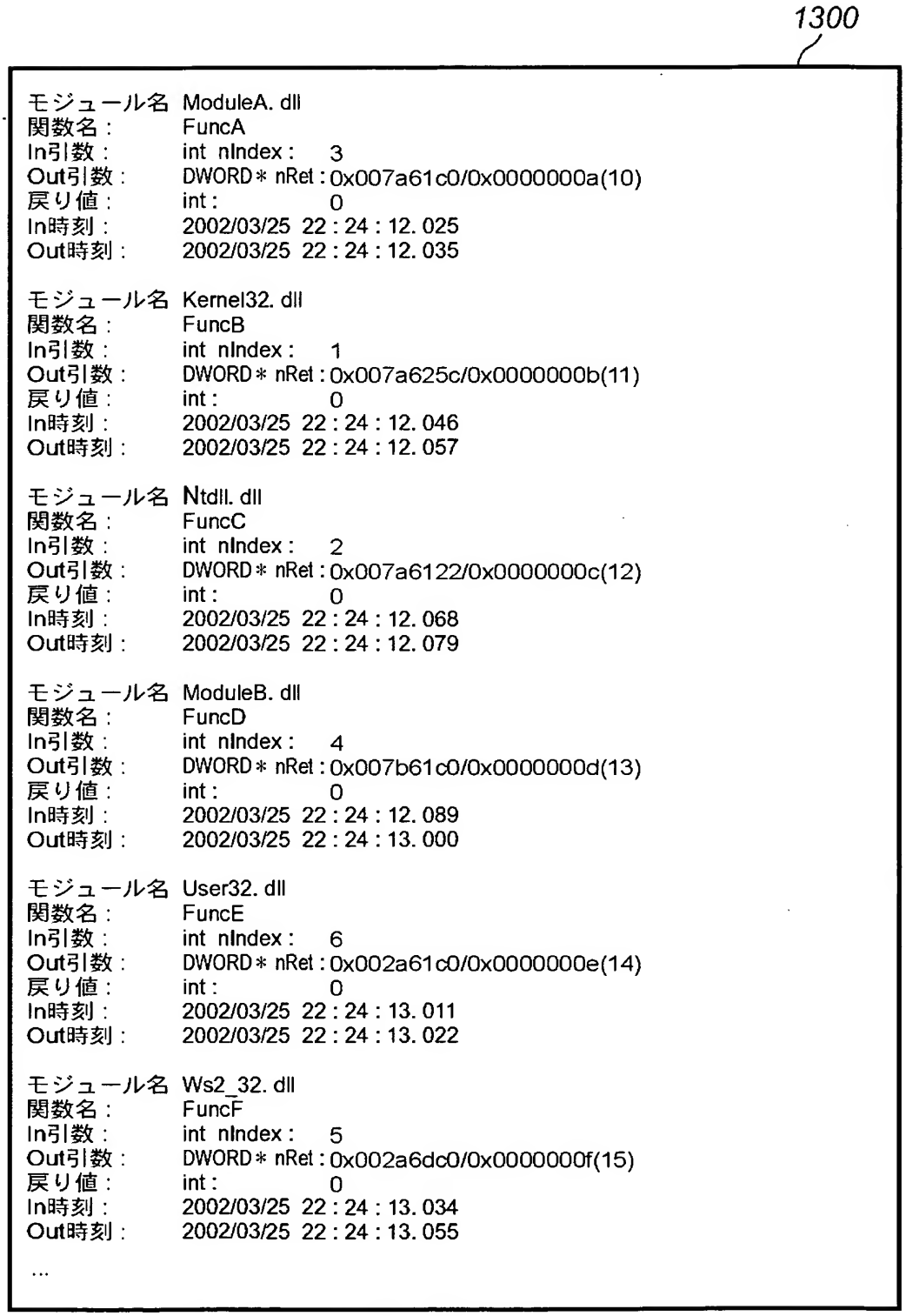
【図１４】


Start
S1400
OSモジュールリストを取得
S1401
呼び出し元モジュールを
OSモジュールリストから検索
S1402
S1403
呼び出し元が
OSモジュール？
YES
NO
S1404
DLL名/Interface名/関数名/Method名
/呼び出し時の時刻をHDDに保存
S1409
OSモジュールリストに
呼び出し先のモジュールを追加
パラメータをHDDに保存
S1405
S1406
元の関数を呼び出す
S1410
元の関数を呼び出す
DLL名/Interface名/関数名/Method名
/終了時の時刻をHDDに保存
S1407
パラメータ/戻り値をHDDに保存
S1408
関数/メソッドのreturn処理
S1411
S1412
終了命令？
NO
YES
End
S1413

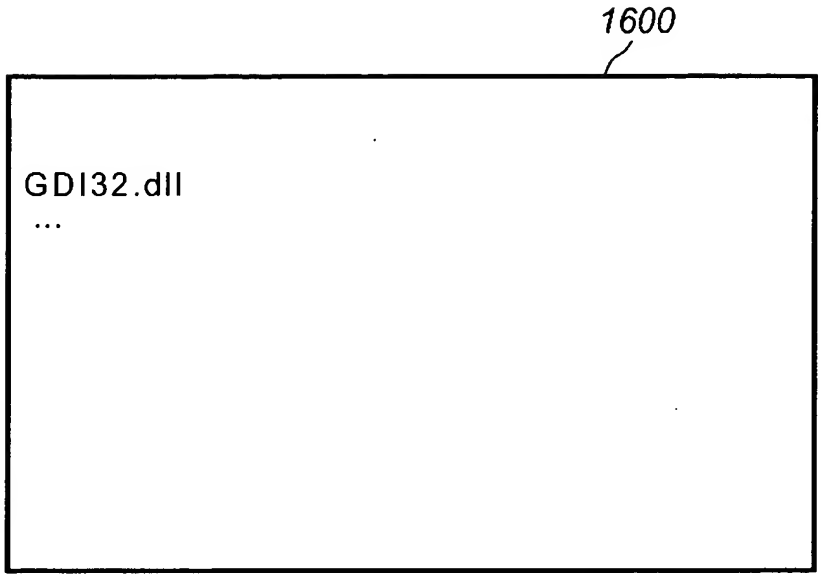
【図１５Ａ】


1500
モジュール名 ModuleA. dll
関数名： FuncA
In引数： int nIndex： 3
Out引数： DWORD＊ nRet：0x007a61c0/0x0000000a(10)
戻り値： int： 0
In時刻： 2002/03/25 22：24：12. 025
Out時刻： 2002/03/25 22：24：12. 035
モジュール名 Kernel32. dll
関数名： FuncB
In引数： int nIndex： 1
Out引数： DWORD＊ nRet：0x007a625c/0x0000000b(11)
戻り値： int： 0
In時刻： 2002/03/25 22：24：12. 046
Out時刻： 2002/03/25 22：24：12. 057
モジュール名 ModuleB. dll
関数名： FuncD
In引数： int nIndex： 4
Out引数： DWORD＊ nRet：0x007b61c0/0x0000000d(13)
戻り値： int： 0
In時刻： 2002/03/25 22：24：12. 089
Out時刻： 2002/03/25 22：24：13. 000
モジュール名 User32. dll
関数名： FuncE
In引数： int nIndex： 6
Out引数： DWORD＊ nRet：0x002a61c0/0x0000000e(14)
戻り値： int： 0
In時刻： 2002/03/25 22：24：13. 011
Out時刻： 2002/03/25 22：24：13. 022
...

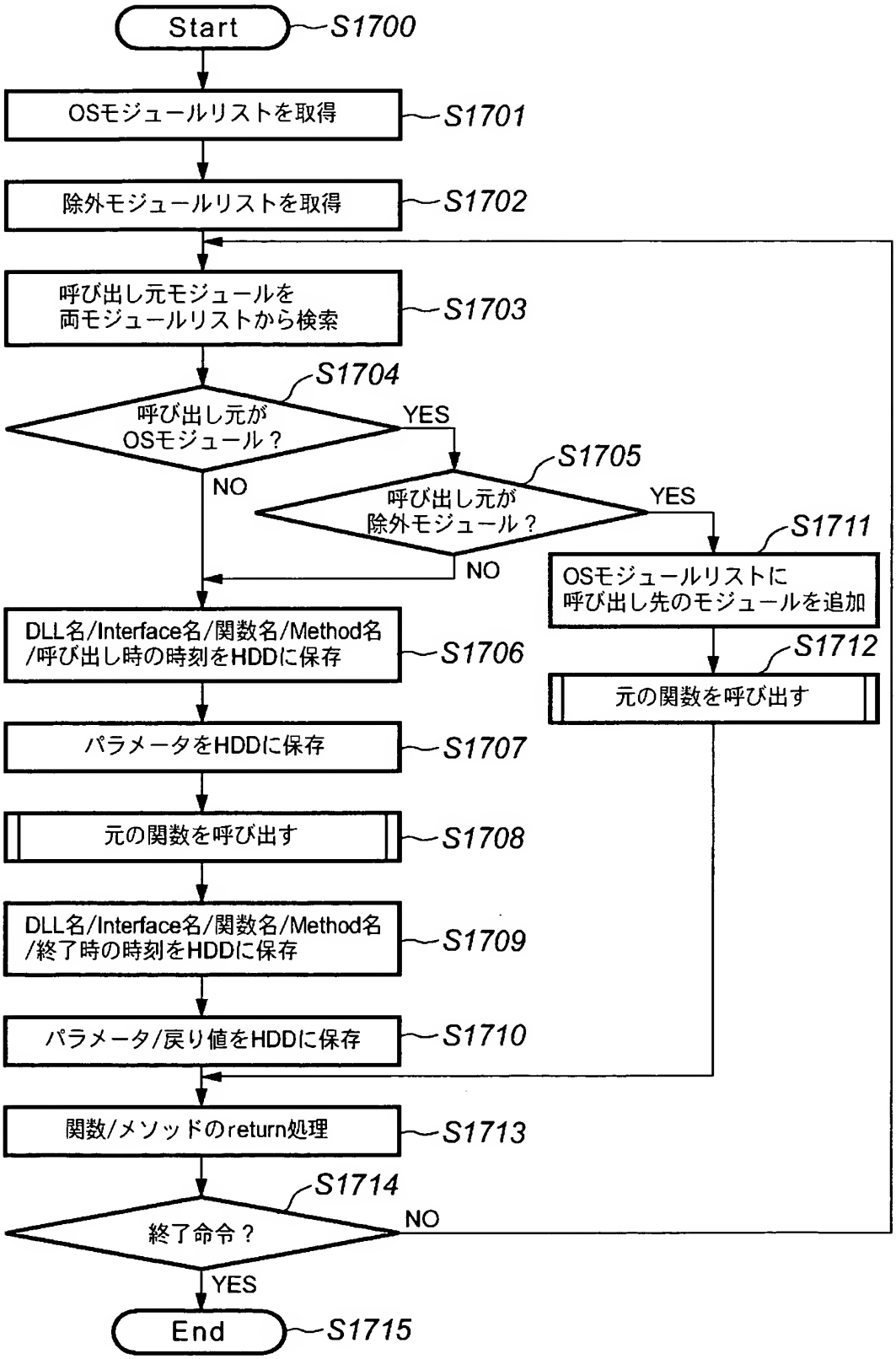
【図１５Ｂ】


1300
モジュール名 ModuleA. dll
関数名： FuncA
In引数： int nIndex： 3
Out引数： DWORD＊ nRet：0x007a61c0/0x0000000a(10)
戻り値： int： 0
In時刻： 2002/03/25 22：24：12. 025
Out時刻： 2002/03/25 22：24：12. 035
モジュール名 Kernel32. dll
関数名： FuncB
In引数： int nIndex： 1
Out引数： DWORD＊ nRet：0x007a625c/0x0000000b(11)
戻り値： int： 0
In時刻： 2002/03/25 22：24：12. 046
Out時刻： 2002/03/25 22：24：12. 057
モジュール名 Ntdll. dll
関数名： FuncC
In引数： int nIndex： 2
Out引数： DWORD＊ nRet：0x007a6122/0x0000000c(12)
戻り値： int： 0
In時刻： 2002/03/25 22：24：12. 068
Out時刻： 2002/03/25 22：24：12. 079
モジュール名 ModuleB. dll
関数名： FuncD
In引数： int nIndex： 4
Out引数： DWORD＊ nRet：0x007b61c0/0x0000000d(13)
戻り値： int： 0
In時刻： 2002/03/25 22：24：12. 089
Out時刻： 2002/03/25 22：24：13. 000
モジュール名 User32. dll
関数名： FuncE
In引数： int nIndex： 6
Out引数： DWORD＊ nRet：0x002a61c0/0x0000000e(14)
戻り値： int： 0
In時刻： 2002/03/25 22：24：13. 011
Out時刻： 2002/03/25 22：24：13. 022
モジュール名 Ws2_32. dll
関数名： FuncF
In引数： int nIndex： 5
Out引数： DWORD＊ nRet：0x002a6dc0/0x0000000f(15)
戻り値： int： 0
In時刻： 2002/03/25 22：24：13. 034
Out時刻： 2002/03/25 22：24：13. 055
...

【図１６】



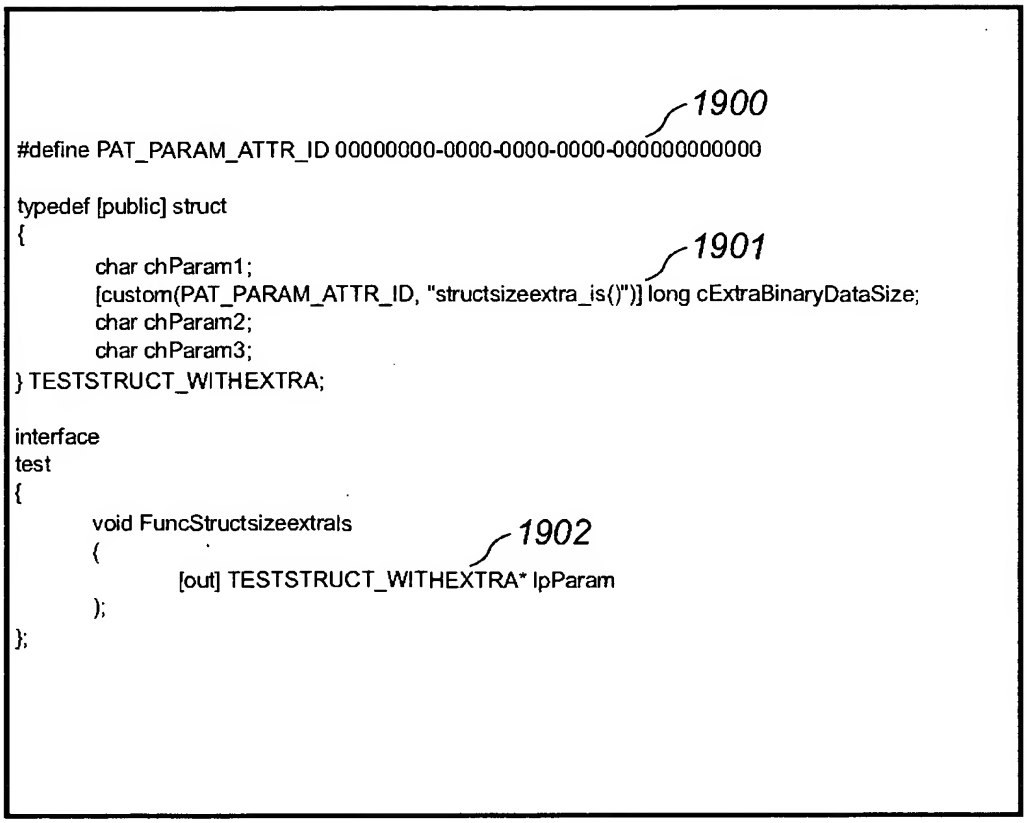
【図１７】


Start
S1700
OSモジュールリストを取得
S1701
除外モジュールリストを取得
S1702
呼び出し元モジュールを
両モジュールリストから検索
S1703
S1704
呼び出し元が
OSモジュール？
YES
NO
S1705
呼び出し元が
除外モジュール？
YES
NO
S1711
OSモジュールリストに
呼び出し先のモジュールを追加
S1712
元の関数を呼び出す
DLL名/Interface名/関数名/Method名
/呼び出し時の時刻をHDDに保存
S1706
パラメータをHDDに保存
S1707
元の関数を呼び出す
S1708
DLL名/Interface名/関数名/Method名
/終了時の時刻をHDDに保存
S1709
パラメータ/戻り値をHDDに保存
S1710
関数/メソッドのreturn処理
S1713
S1714
終了命令？
NO
YES
End
S1715

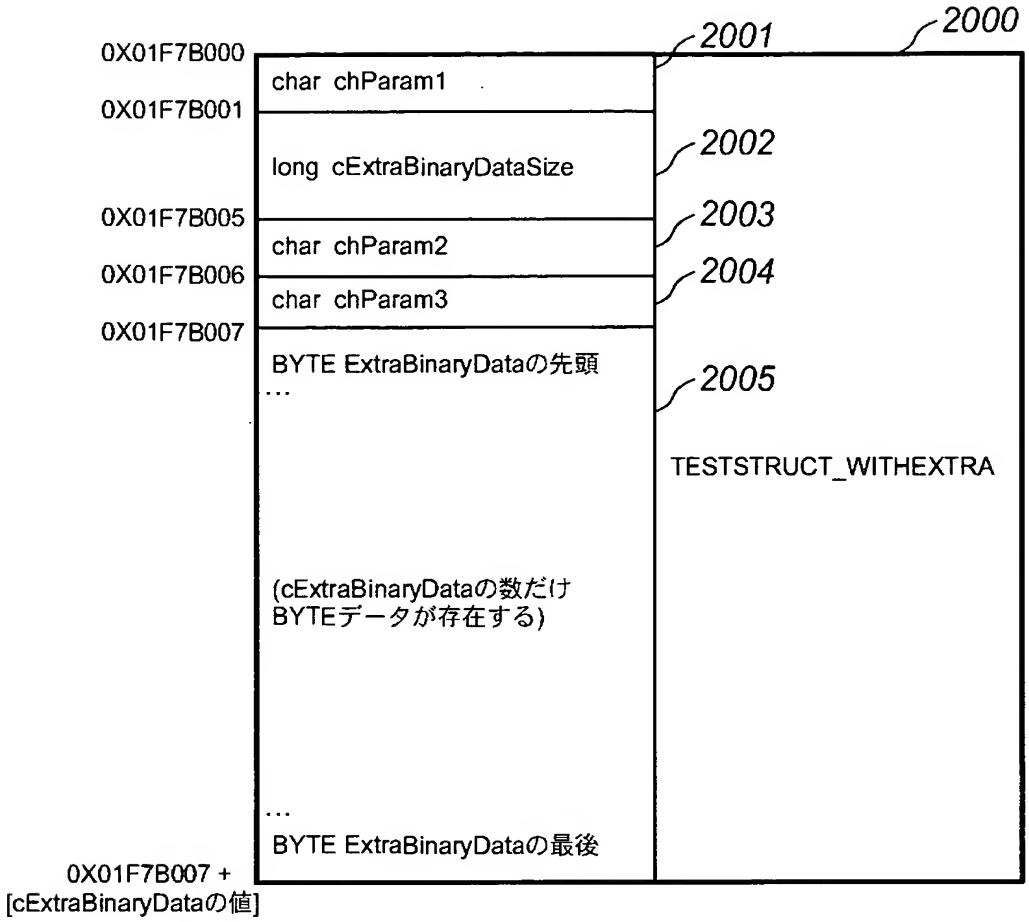
【図１８】

```
[
   uuid(58DB5633-0694-4340-97CE-4E1AC6BFFBA7),      / /TestDllStd.
   helpstring("TestDllStd Type Library For PAT"),
   version(1.0)
]

library TestDllStd
{
        typedef [public] struct
        {
                char chParam;
                unsigned char uchParam;
                short sParam;
                unsigned short usParam;
                int nParam;
                unsigned int unParam;
                long lParam;
                unsigned long ulParam;
                double dbParam;
                float fParam;
        } TESTSTRUCT;
        typedef [public] TESTSTRUCT *LPTESTSTRUCT;

/ /DEFINE_GUID(GUID_PROGID, 0x8e037d65, 0xefa0, 0x40e7, 0x91, 0x43, 0xef,0x70, 0x56, 0x94, 0x5b,
0x79);
  [
    uuid(8E037D65-EFAO-40e7-9143-EF7056945B79),
    helpstring("TestDllStd.dll for PAT object." ),
  ]
        interface
        test
        {
                char _stdcall FuncCharStd([in] char chParam);
                char* _stdcall FuncPCharStd([in, out] char* lpchParam);

                TESTSTRUCT _stdcall FuncStructStd([in]TESTSTRUCT TestStruct);
                LPTESTSTRUCT _stdcall FuncPStructStd([in, out]LPTESTSTRUCT lpTestStruct);
        };
}
```

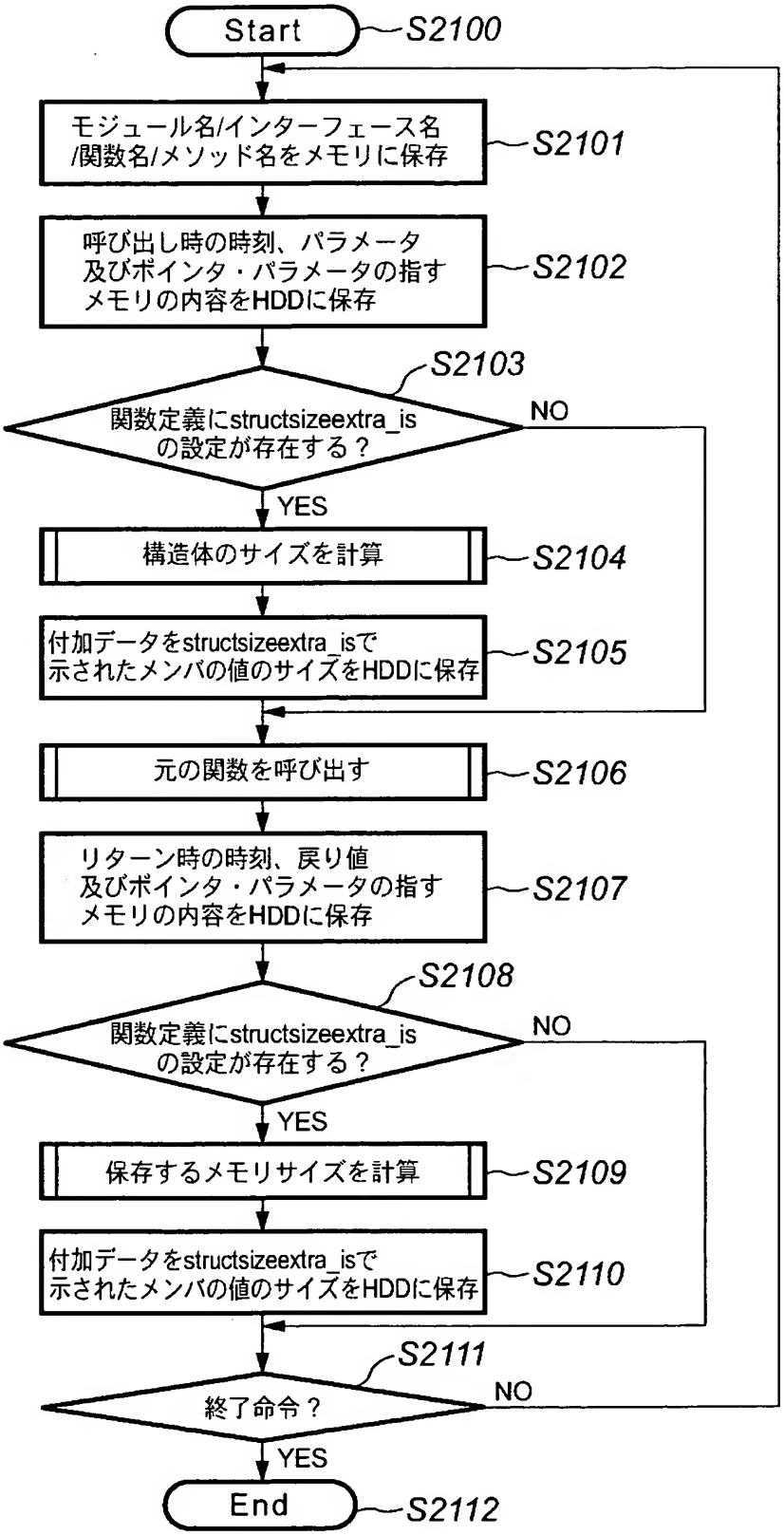
【図１９】


1900
#define PAT_PARAM_ATTR_ID 00000000-0000-0000-0000-000000000000
typedef [public] struct
{
char chParam1;
1901
[custom(PAT_PARAM_ATTR_ID, "structsizeextra_is()")] long cExtraBinaryDataSize;
char chParam2;
char chParam3;
} TESTSTRUCT_WITHEXTRA;
interface
test
{
void FuncStructsizeextrals
1902
(
[out] TESTSTRUCT_WITHEXTRA* lpParam
);
};

【図２０】


2000
2001
0X01F7B000
char chParam1
0X01F7B001
long cExtraBinaryDataSize
2002
0X01F7B005
2003
char chParam2
0X01F7B006
2004
char chParam3
0X01F7B007
BYTE ExtraBinaryDataの先頭
...
2005
TESTSTRUCT_WITHEXTRA
(cExtraBinaryDataの数だけ
BYTEデータが存在する)
...
BYTE ExtraBinaryDataの最後
0X01F7B007 +
[cExtraBinaryDataの値]

【図２１】


Start
S2100
モジュール名/インターフェース名
/関数名/メソッド名をメモリに保存
S2101
呼び出し時の時刻、パラメータ
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S2102
S2103
関数定義にstructsizeextra_is
の設定が存在する？
NO
YES
構造体のサイズを計算
S2104
付加データをstructsizeextra_isで
示されたメンバの値のサイズをHDDに保存
S2105
元の関数を呼び出す
S2106
リターン時の時刻、戻り値
及びポインタ・パラメータの指す
メモリの内容をHDDに保存
S2107
S2108
関数定義にstructsizeextra_is
の設定が存在する？
NO
YES
保存するメモリサイズを計算
S2109
付加データをstructsizeextra_isで
示されたメンバの値のサイズをHDDに保存
S2110
S2111
終了命令？
NO
YES
End
S2112

【図２２】

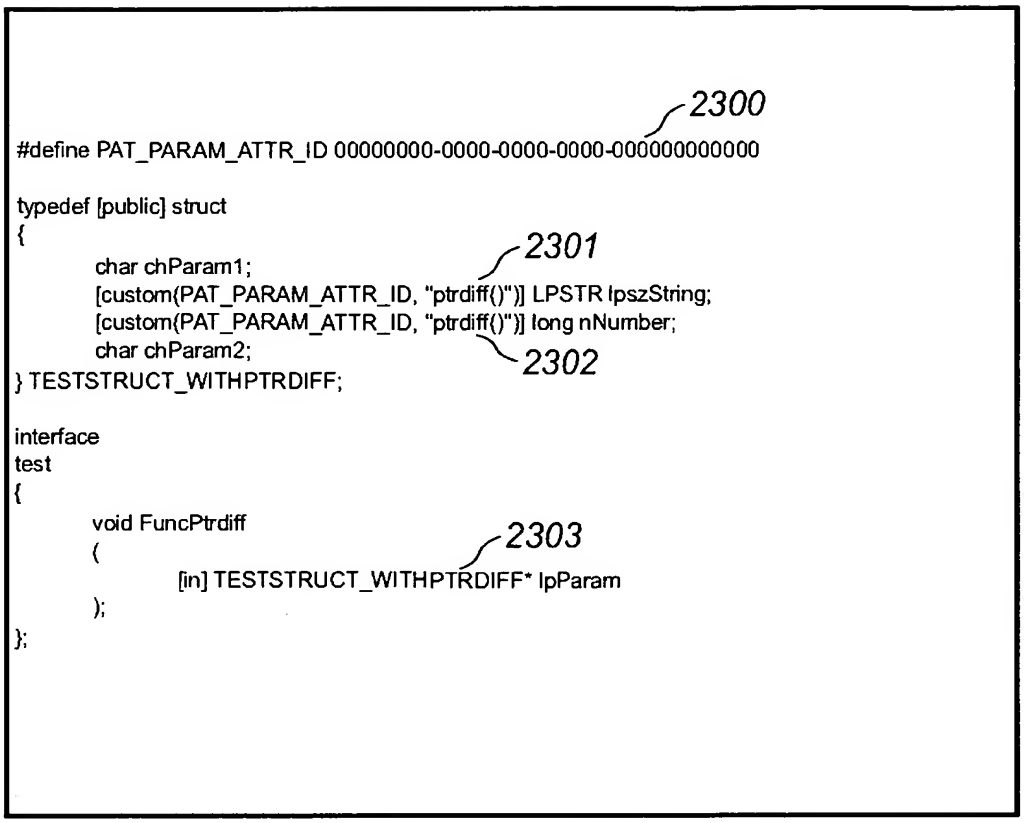
2200

```
モジュール名：    TestDllStd.DLL
関数名：          Func Structsizeextrals
引数(in)：
引数(out)：       struct TESTSTRUCT_WITHEXTRA：
                    char chParam1 ："a"
                    long cExtraBinaryDataSize: 8
                    char chParam2 ："b"
                    char chParam3 ："c"
                    ExtraBinaryData：0x01F7B007. Data ID=0x0001
戻り値：          void：
In時刻：          2002/03/25 22：24：12.025
Out時刻：         2002/03/25 22：24：12.035

モジュール名：    TestDllStd.DLL
関数名：          Func Structsizeextrals
引数(in)：
引数(out)：       struct TESTSTRUCT_WITHEXTRA：
                    char chParam1 ："d"
                    long cExtraBinaryDataSize: 40
                    char chParam2 ："e"
                    char chParam3 ："f"
                    ExtraBinaryData：0x01F7B007. Data ID=0x0002
戻り値：          void：
In時刻：          2002/03/25 22：24：12.046
Out時刻：         2002/03/25 22：24：12.057

モジュール名：    TestDllStd.DLL
関数名：          Func Structsizeextrals
引数(in)：
引数(out)：       struct TESTSTRUCT_WITHEXTRA：
                    char chParam1 ："g"
                    long cExtraBinaryDataSize: 5
                    char chParam2 ："h"
                    char chParam3 ："i"
                    ExtraBinaryData：0x01F7B007. Data ID=0x0003
戻り値：          void：
In時刻：          2002/03/25 22：24：12.068
Out時刻：         2002/03/25 22：24：12.079

モジュール名：    TestDllStd.DLL
関数名：          Func Structsizeextrals
引数(in)：
引数(out)：       struct TESTSTRUCT_WITHEXTRA：
                    char chParam1 ："j"
                    long cExtraBinaryDataSize: 7
                    char chParam2 ："k"
                    char chParam3 ："l"
                    ExtraBinaryData：0x01F7B007. Data ID=0x0004
戻り値：          void：
In時刻：          2002/03/25 22：24：12.100
Out時刻：         2002/03/25 22：24：12.179

・・・
```

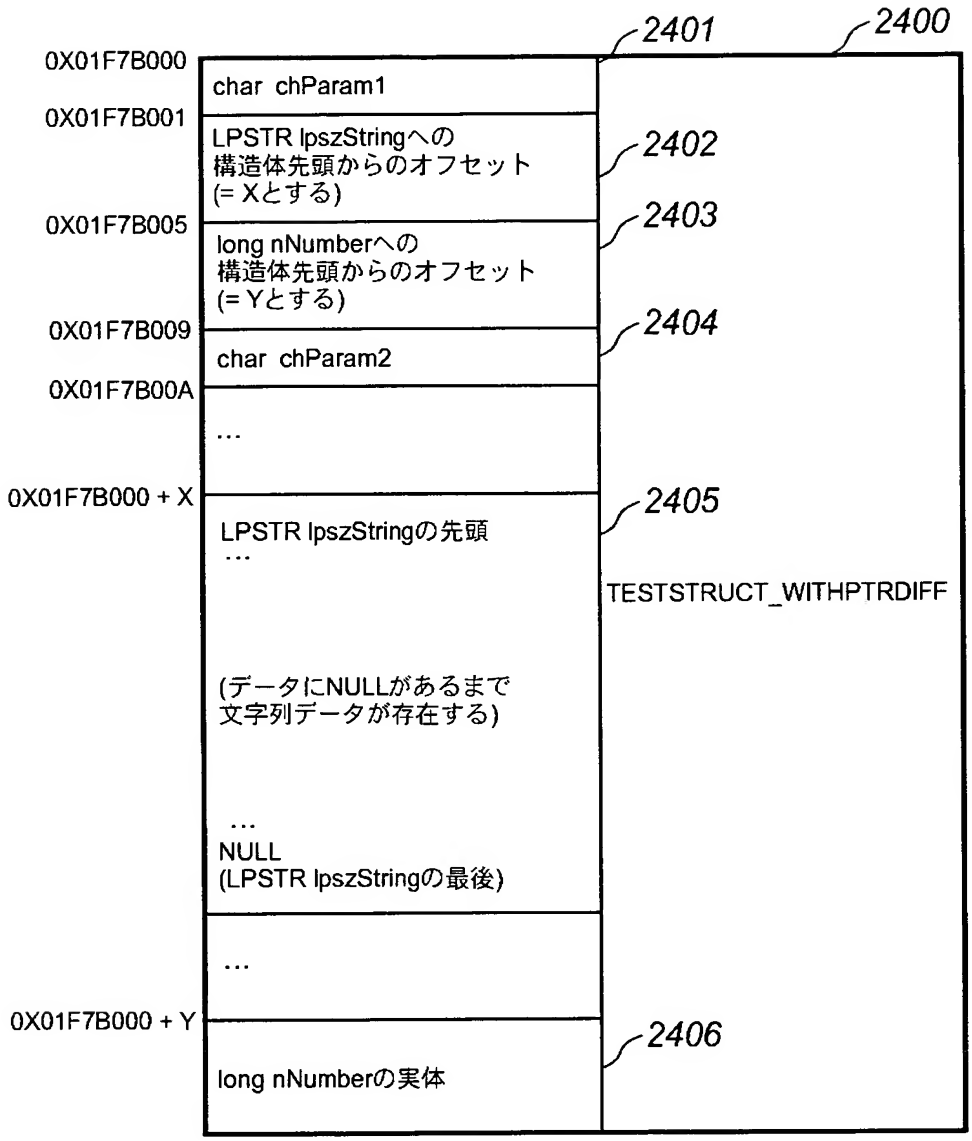
2201

```
Data ID: 0x0001
Size: 8

00000000: 10 00 00 00 4A 03 A5 20

Data ID: 0x0002
Size: 40

00000000: 05 00 00 00 4A 03 A5 20
00000008: 05 00 00 00 4B 03 A5 20
00000010: 05 00 00 00 4C 03 A5 20
00000018: 05 00 00 00 4D 03 A5 20
00000020: 05 00 00 00 4E 03 A5 20

Data ID: 0x0003
Size: 40

00000000: 66 4A 70 50 00

Data ID: 0x0004
Size: 7

00000000: 01 5D 66 B2 20 49 20

 ...
```

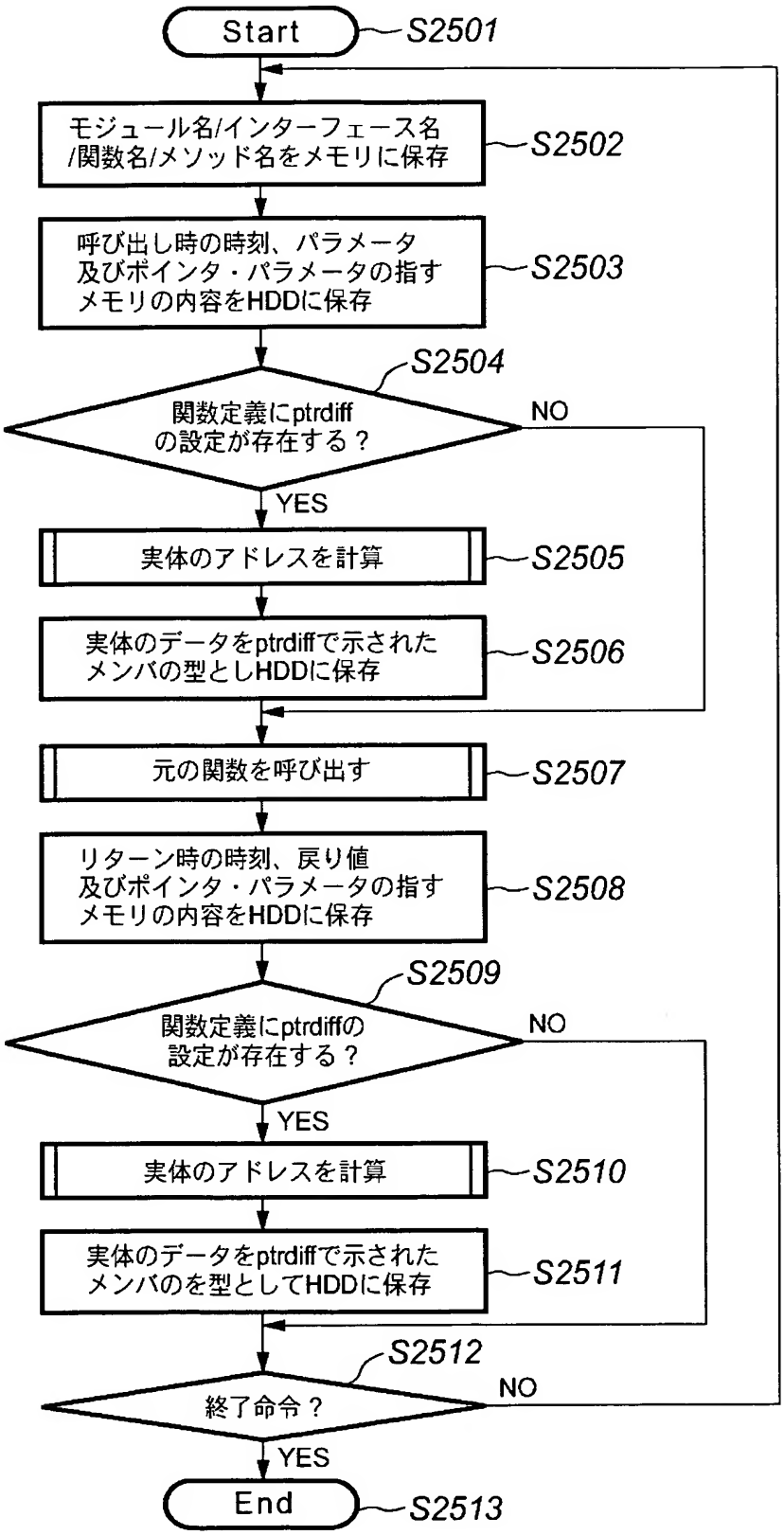
【図２３】


2300
#define PAT_PARAM_ATTR_ID 00000000-0000-0000-0000-000000000000
typedef [public] struct
{
char chParam1;
2301
[custom(PAT_PARAM_ATTR_ID, "ptrdiff()")] LPSTR lpszString;
[custom(PAT_PARAM_ATTR_ID, "ptrdiff()")] long nNumber;
char chParam2;
2302
} TESTSTRUCT_WITHPTRDIFF;
interface
test
{
void FuncPtrdiff
(
2303
[in] TESTSTRUCT_WITHPTRDIFF* lpParam
);
};

【図２４】


2401
2400
0X01F7B000
char chParam1
0X01F7B001
LPSTR lpszStringへの
構造体先頭からのオフセット
(= Xとする)
2402
0X01F7B005
long nNumberへの
構造体先頭からのオフセット
(= Yとする)
2403
0X01F7B009
char chParam2
2404
0X01F7B00A
…
0X01F7B000 + X
LPSTR lpszStringの先頭
…
2405
TESTSTRUCT_WITHPTRDIFF
(データにNULLがあるまで
文字列データが存在する)
…
NULL
(LPSTR lpszStringの最後)
…
0X01F7B000 + Y
long nNumberの実体
2406

【図２５】

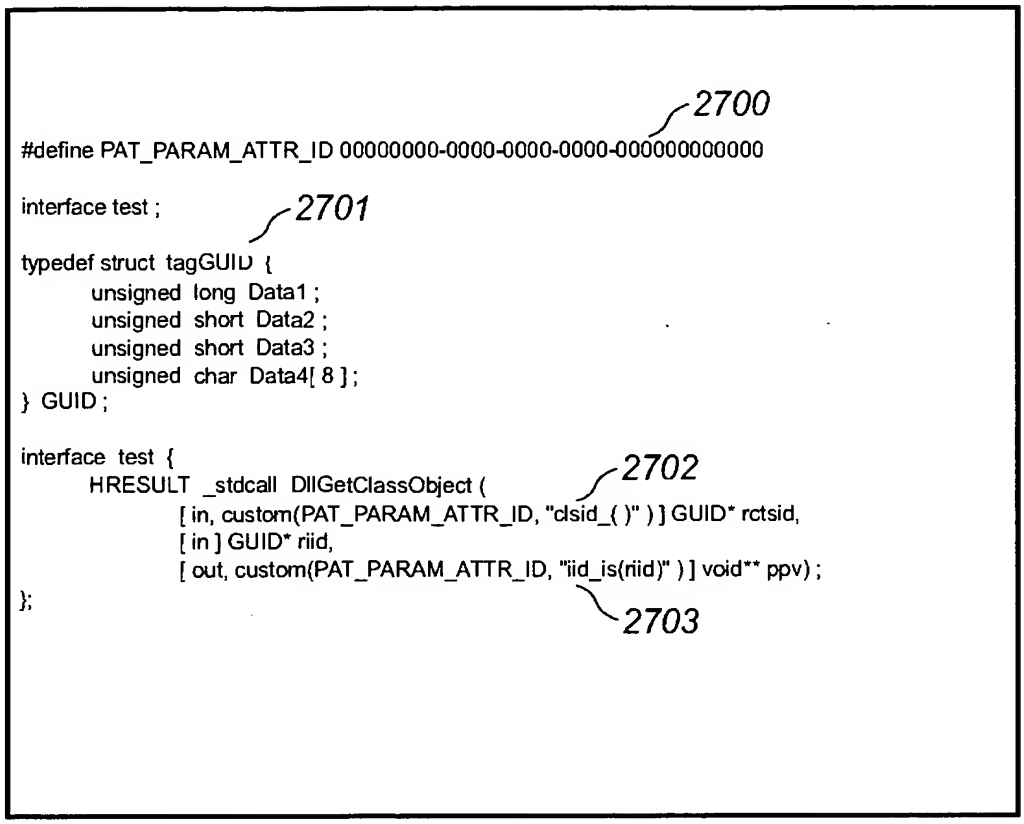

Start
S2501
モジュール名/インターフェース名/関数名/メソッド名をメモリに保存
S2502
呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をHDDに保存
S2503
S2504
関数定義にptrdiffの設定が存在する？
NO
YES
実体のアドレスを計算
S2505
実体のデータをptrdiffで示されたメンバの型としHDDに保存
S2506
元の関数を呼び出す
S2507
リターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内容をHDDに保存
S2508
S2509
関数定義にptrdiffの設定が存在する？
NO
YES
実体のアドレスを計算
S2510
実体のデータをptrdiffで示されたメンバのを型としてHDDに保存
S2511
S2512
終了命令？
NO
YES
End
S2513

【図２６】

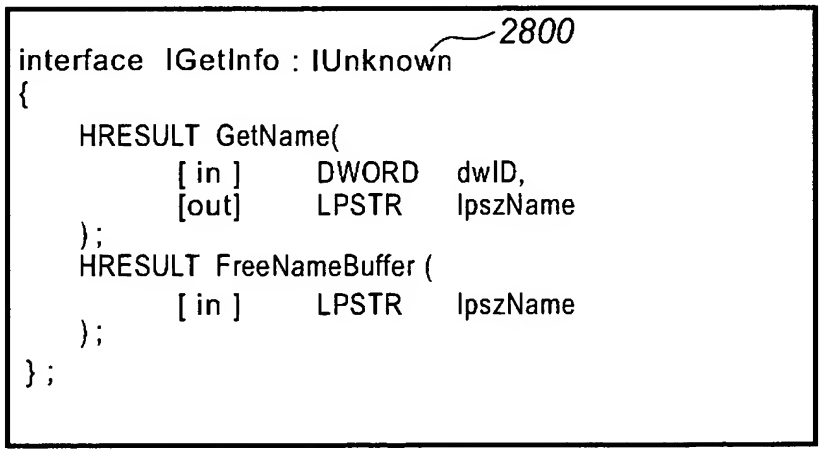
2600

```
モジュール名 :  TestDllStd.DLL
関数名 :        FuncPtrdiff
引数(in) :      struct TESTSTRUCT_WITHPTRDIFF :
                  char  chParam1  : " a"
                  Offset to LPSTR lpszString :  16
                  LPSTR lpszString :  "Test1"
                  Offset to long nNumber :  28
                  long nNumber :  1
                  char  chParam3  : " b"
引数(out) :
戻り値 :        void :
In時刻 :        2002/03/25  22 : 24 : 12. 025
Out時刻 :       2002/03/25  22 : 24 : 12. 035

モジュール名 :  TestDllStd.DLL
関数名 :        FuncPtrdiff
引数(in) :      struct TESTSTRUCT_WITHPTRDIFF :
                  char  chParam1  : " c"
                  Offset to LPSTR lpszString :  18
                  LPSTR lpszString :  "Test2"
                  Offset to long nNumber :  30
                  long nNumber :  2
                  char  chParam3  : " d"
引数(out) :
戻り値 :        void :
In時刻 :        2002/03/25  22 : 24 : 12. 046
Out時刻 :       2002/03/25  22 : 24 : 12. 057

モジュール名 :  TestDllStd.DLL
関数名 :        FuncPtrdiff
引数(in) :      struct TESTSTRUCT_WITHPTRDIFF :
                  char  chParam1  : " e"
                  Offset to LPSTR lpszString :  20
                  LPSTR lpszString :  "Test3"
                  Offset to long nNumber :  32
                  long nNumber :  3
                  char  chParam3  : " f"
引数(out) :
戻り値 :        void :
In時刻 :        2002/03/25  22 : 24 : 12. 068
Out時刻 :       2002/03/25  22 : 24 : 12. 079

モジュール名 :  TestDllStd.DLL
関数名 :        FuncPtrdiff
引数(in) :      struct TESTSTRUCT_WITHPTRDIFF :
                  char  chParam1  : "g"
                  Offset to LPSTR lpszString :  16
                  LPSTR lpszString :  "Test4"
                  Offset to long nNumber :  28
                  long nNumber :  29
                  char  chParam3  : " h"
引数(out) :
戻り値 :        void :
In時刻 :        2002/03/25  22 : 24 : 12. 100
Out時刻 :       2002/03/25  22 : 24 : 12. 179


・・・
```

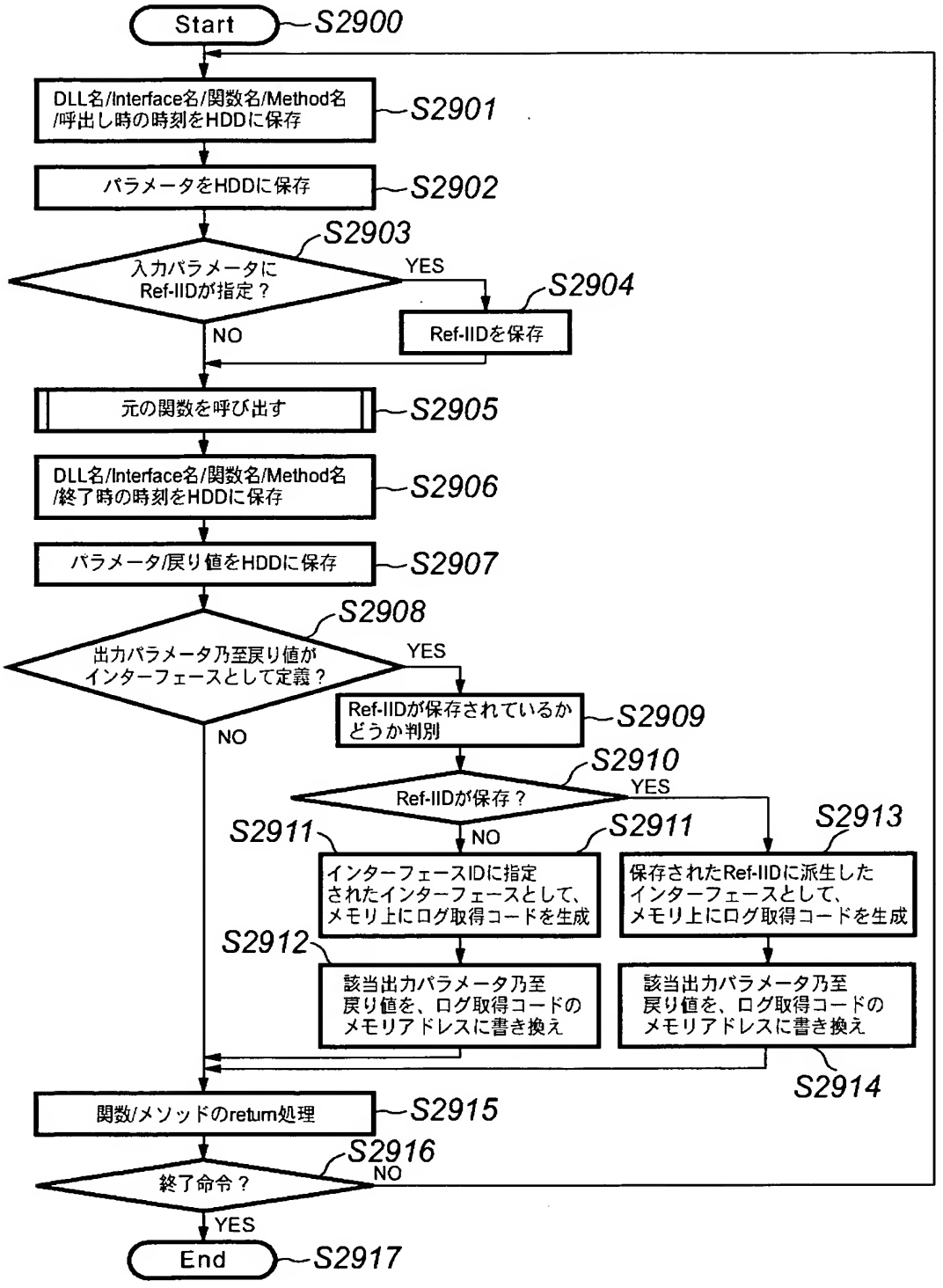
【図２７】


2700
#define PAT_PARAM_ATTR_ID 00000000-0000-0000-0000-000000000000
interface test ;
2701
typedef struct tagGUID {
unsigned long Data1 ;
unsigned short Data2 ;
unsigned short Data3 ;
unsigned char Data4[8] ;
} GUID ;
interface test {
HRESULT _stdcall DllGetClassObject (
2702
[in, custom(PAT_PARAM_ATTR_ID, "clsid_( )")] GUID* rctsid,
[in] GUID* riid,
[out, custom(PAT_PARAM_ATTR_ID, "iid_is(riid)")] void** ppv) ;
};
2703

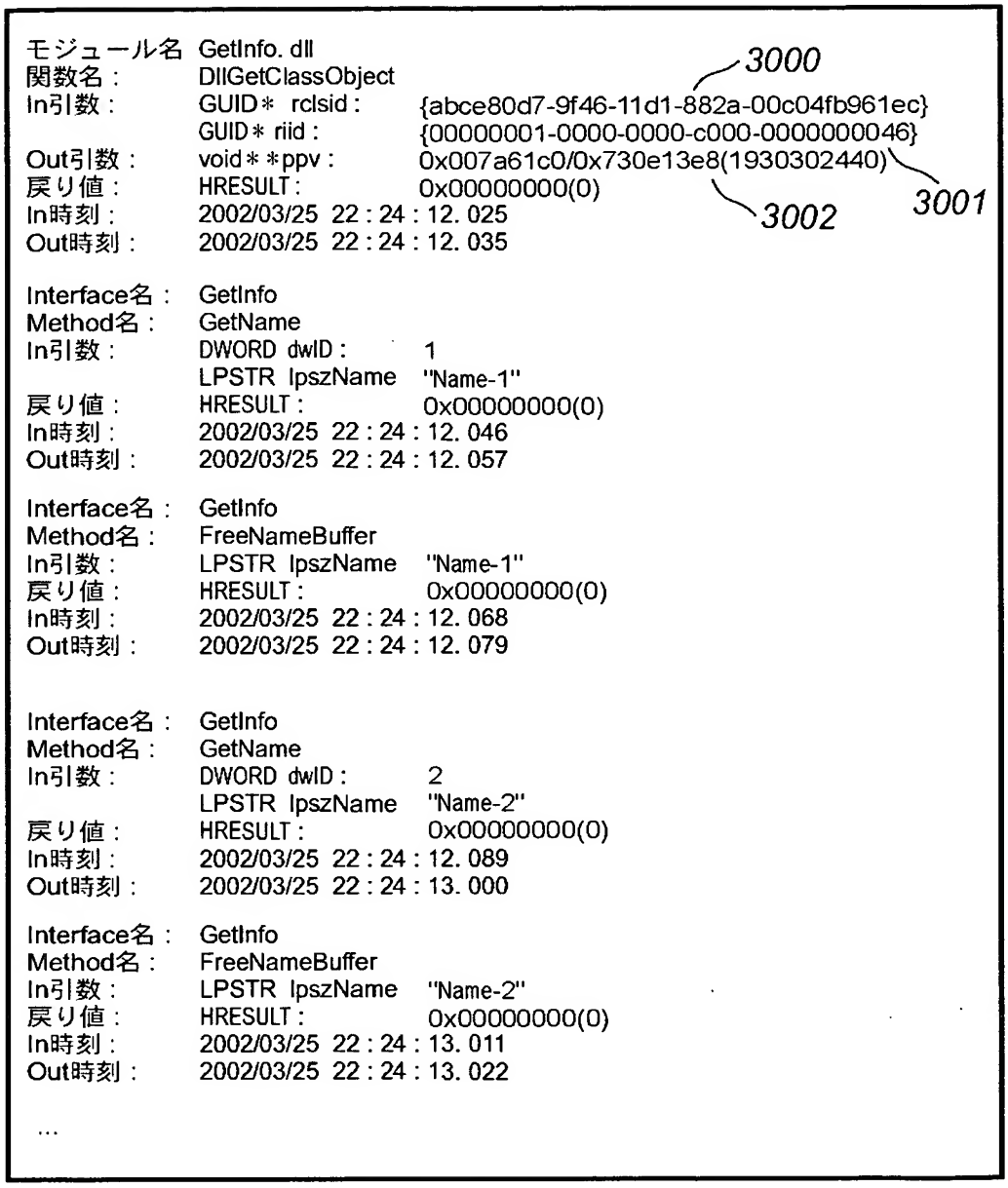
【図２８】


2800
interface IGetInfo : IUnknown
{
HRESULT GetName(
[in] DWORD dwID,
[out] LPSTR lpszName
) ;
HRESULT FreeNameBuffer (
[in] LPSTR lpszName
) ;
} ;

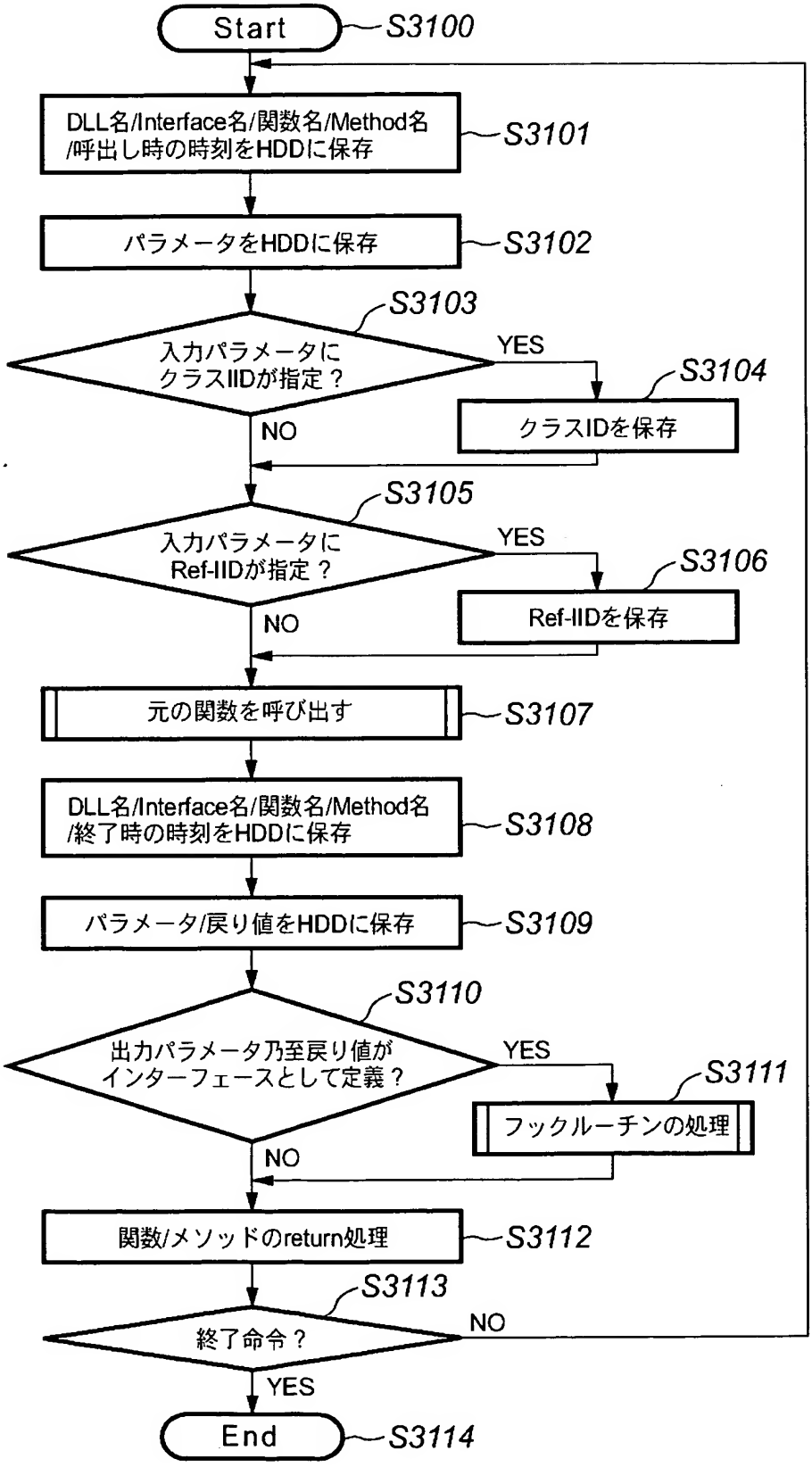
【図２９】


Start
S2900
DLL名/Interface名/関数名/Method名/呼出し時の時刻をHDDに保存
S2901
パラメータをHDDに保存
S2902
S2903
入力パラメータにRef-IIDが指定？
YES
S2904
Ref-IIDを保存
NO
元の関数を呼び出す
S2905
DLL名/Interface名/関数名/Method名/終了時の時刻をHDDに保存
S2906
パラメータ/戻り値をHDDに保存
S2907
S2908
出力パラメータ乃至戻り値がインターフェースとして定義？
YES
Ref-IIDが保存されているかどうか判別
S2909
NO
S2910
Ref-IIDが保存？
YES
NO
S2911
S2911
S2913
インターフェースIDに指定されたインターフェースとして、メモリ上にログ取得コードを生成
保存されたRef-IIDに派生したインターフェースとして、メモリ上にログ取得コードを生成
S2912
該当出力パラメータ乃至戻り値を、ログ取得コードのメモリアドレスに書き換え
該当出力パラメータ乃至戻り値を、ログ取得コードのメモリアドレスに書き換え
S2914
関数/メソッドのreturn処理
S2915
S2916
終了命令？
NO
YES
End
S2917

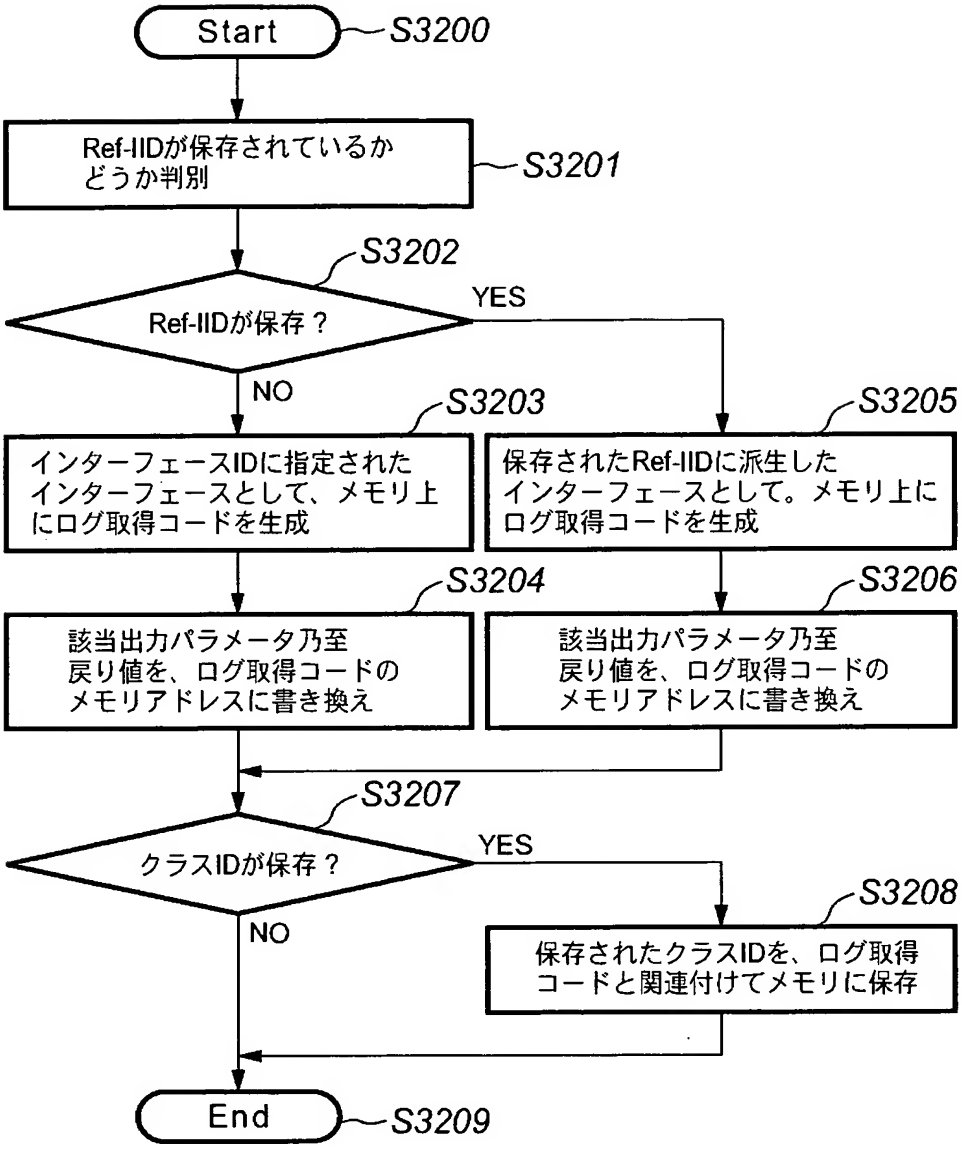
【図３０】


モジュール名 GetInfo. dll
関数名： DllGetClassObject
In引数： GUID＊ rclsid： {abce80d7-9f46-11d1-882a-00c04fb961ec}
GUID＊ riid： {00000001-0000-0000-c000-0000000046}
Out引数： void＊＊ppv： 0x007a61c0/0x730e13e8(1930302440)
戻り値： HRESULT： 0x00000000(0)
In時刻： 2002/03/25 22：24：12. 025
Out時刻： 2002/03/25 22：24：12. 035
Interface名： GetInfo
Method名： GetName
In引数： DWORD dwID： 1
LPSTR lpszName "Name-1"
戻り値： HRESULT： 0x00000000(0)
In時刻： 2002/03/25 22：24：12. 046
Out時刻： 2002/03/25 22：24：12. 057
Interface名： GetInfo
Method名： FreeNameBuffer
In引数： LPSTR lpszName "Name-1"
戻り値： HRESULT： 0x00000000(0)
In時刻： 2002/03/25 22：24：12. 068
Out時刻： 2002/03/25 22：24：12. 079
Interface名： GetInfo
Method名： GetName
In引数： DWORD dwID： 2
LPSTR lpszName "Name-2"
戻り値： HRESULT： 0x00000000(0)
In時刻： 2002/03/25 22：24：12. 089
Out時刻： 2002/03/25 22：24：13. 000
Interface名： GetInfo
Method名： FreeNameBuffer
In引数： LPSTR lpszName "Name-2"
戻り値： HRESULT： 0x00000000(0)
In時刻： 2002/03/25 22：24：13. 011
Out時刻： 2002/03/25 22：24：13. 022
...
3000
3001
3002

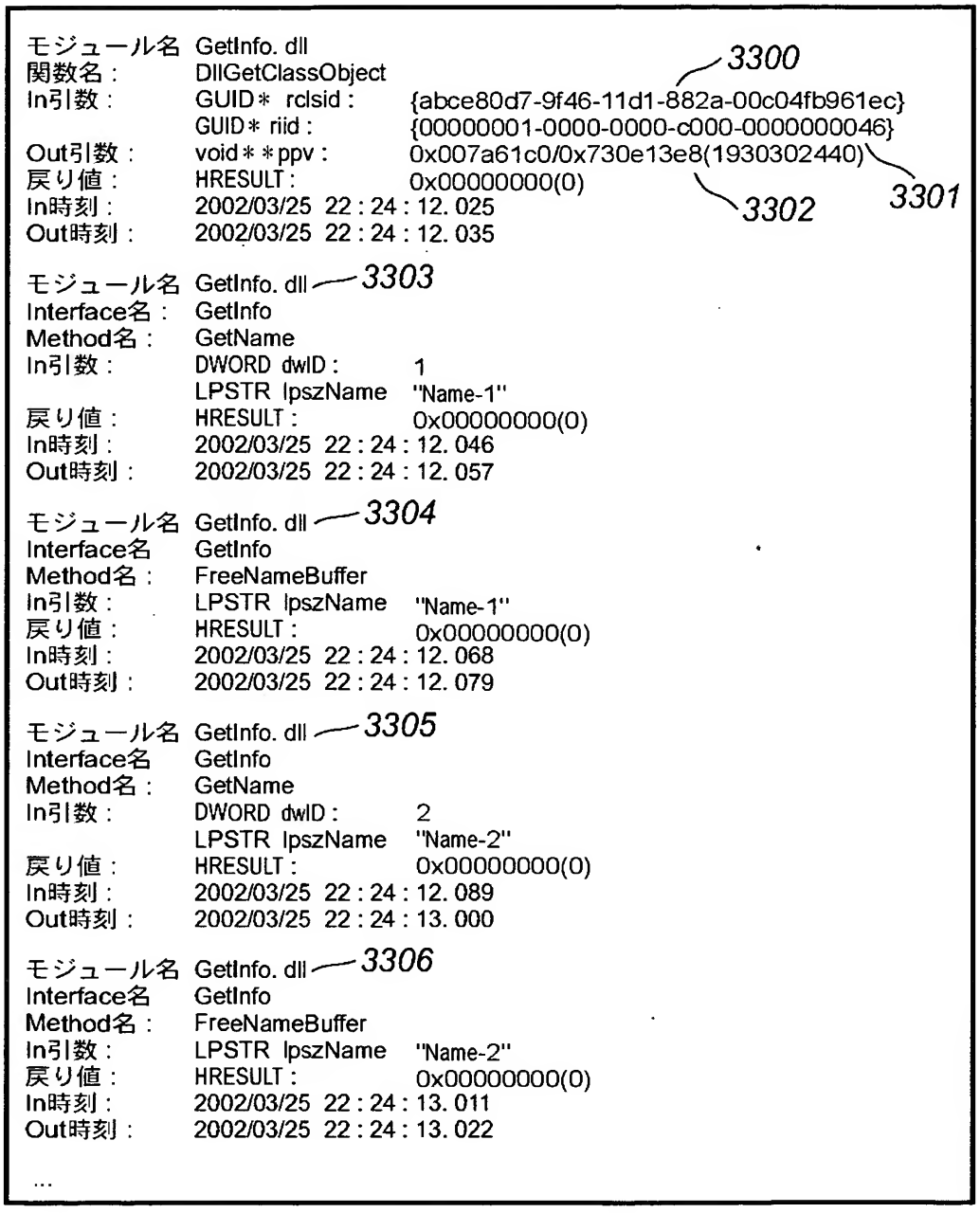
【図３１】


Start
S3100
DLL名/Interface名/関数名/Method名
/呼出し時の時刻をHDDに保存
S3101
パラメータをHDDに保存
S3102
S3103
入力パラメータに
クラスIIDが指定？
YES
S3104
クラスIDを保存
NO
S3105
入力パラメータに
Ref-IIDが指定？
YES
S3106
Ref-IIDを保存
NO
元の関数を呼び出す
S3107
DLL名/Interface名/関数名/Method名
/終了時の時刻をHDDに保存
S3108
パラメータ/戻り値をHDDに保存
S3109
S3110
出力パラメータ乃至戻り値が
インターフェースとして定義？
YES
S3111
フックルーチンの処理
NO
関数/メソッドのreturn処理
S3112
S3113
終了命令？
NO
YES
End
S3114

【図３２】


Start
S3200
Ref-IIDが保存されているか
どうか判別
S3201
S3202
Ref-IIDが保存？
YES
NO
S3203
インターフェースIDに指定された
インターフェースとして、メモリ上
にログ取得コードを生成
S3205
保存されたRef-IIDに派生した
インターフェースとして。メモリ上に
ログ取得コードを生成
S3204
該当出力パラメータ乃至
戻り値を、ログ取得コードの
メモリアドレスに書き換え
S3206
該当出力パラメータ乃至
戻り値を、ログ取得コードの
メモリアドレスに書き換え
S3207
クラスIDが保存？
YES
NO
S3208
保存されたクラスIDを、ログ取得
コードと関連付けてメモリに保存
End
S3209

【図３３】


モジュール名 GetInfo. dll
関数名： DllGetClassObject
In引数： GUID＊ rclsid： {abce80d7-9f46-11d1-882a-00c04fb961ec}
GUID＊ riid： {00000001-0000-0000-c000-0000000046}
Out引数： void＊＊ppv： 0x007a61c0/0x730e13e8(1930302440)
戻り値： HRESULT： 0x00000000(0)
In時刻： 2002/03/25 22：24：12. 025
Out時刻： 2002/03/25 22：24：12. 035
3300
3301
3302
モジュール名 GetInfo. dll
Interface名： GetInfo
Method名： GetName
In引数： DWORD dwID： 1
LPSTR lpszName "Name-1"
戻り値： HRESULT： 0x00000000(0)
In時刻： 2002/03/25 22：24：12. 046
Out時刻： 2002/03/25 22：24：12. 057
3303
モジュール名 GetInfo. dll
Interface名 GetInfo
Method名： FreeNameBuffer
In引数： LPSTR lpszName "Name-1"
戻り値： HRESULT： 0x00000000(0)
In時刻： 2002/03/25 22：24：12. 068
Out時刻： 2002/03/25 22：24：12. 079
3304
モジュール名 GetInfo. dll
Interface名 GetInfo
Method名： GetName
In引数： DWORD dwID： 2
LPSTR lpszName "Name-2"
戻り値： HRESULT： 0x00000000(0)
In時刻： 2002/03/25 22：24：12. 089
Out時刻： 2002/03/25 22：24：13. 000
3305
モジュール名 GetInfo. dll
Interface名 GetInfo
Method名： FreeNameBuffer
In引数： LPSTR lpszName "Name-2"
戻り値： HRESULT： 0x00000000(0)
In時刻： 2002/03/25 22：24：13. 011
Out時刻： 2002/03/25 22：24：13. 022
3306
...

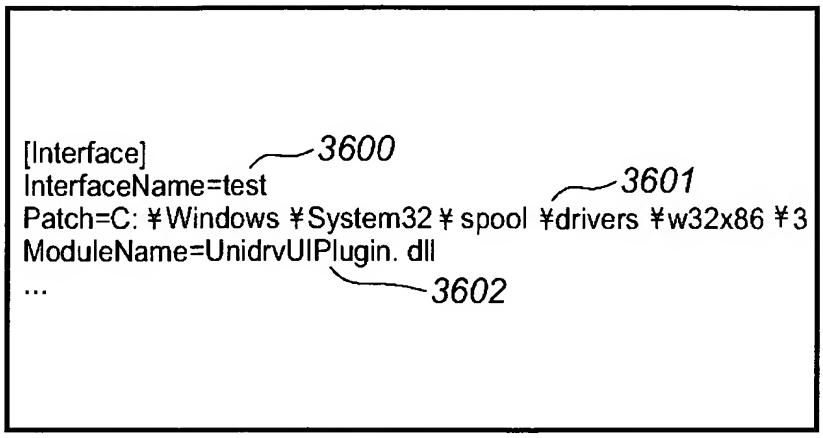
【図３４】

```
[
   uuid(58DB5633-0694-4340-97CE-4E1AC6BFFBA7),     //TestDllStd.
   helpstring("TestDllStd Type Library For PAT"),
   version(1.0)
]

library TestDllStd
{
        typedef [public] struct
        {
                char chParam;
                unsigned char uchParam;
                short sParam;
                unsigned short usParam;
                int nParam;
                unsigned int unParam;
                long lParam;
                unsigned long ulParam;
                double dbParam;
                float fParam;
        } TESTSTRUCT;
        typedef [public] TESTSTRUCT *LPTESTSTRUCT;

//DEFINE_GUID(GUID_PROGID, 0x8e037d65, 0xefa0, 0x40e7, 0x91, 0x43, 0xef, 0x70, 0x56, 0x94, 0
0x79);
  [
    uuid(8E037D65-EFA0-40e7-9143-EF7056945B79),
    helpstring("TestDllStd.dll for PAT object." ),
  ]
        interface
        test
        {
                char _stdcall FuncCharStd([in] char chParam);
                char* _stdcall FuncPCharStd([in, out] char* lpchParam);

                TESTSTRUCT _stdcall FuncStructStd([in]TESTSTRUCT TestStruct);
                LPTESTSTRUCT _stdcall FuncPStructStd([in, out]LPTESTSTRUCT lpTestStruct);
        };
}
```

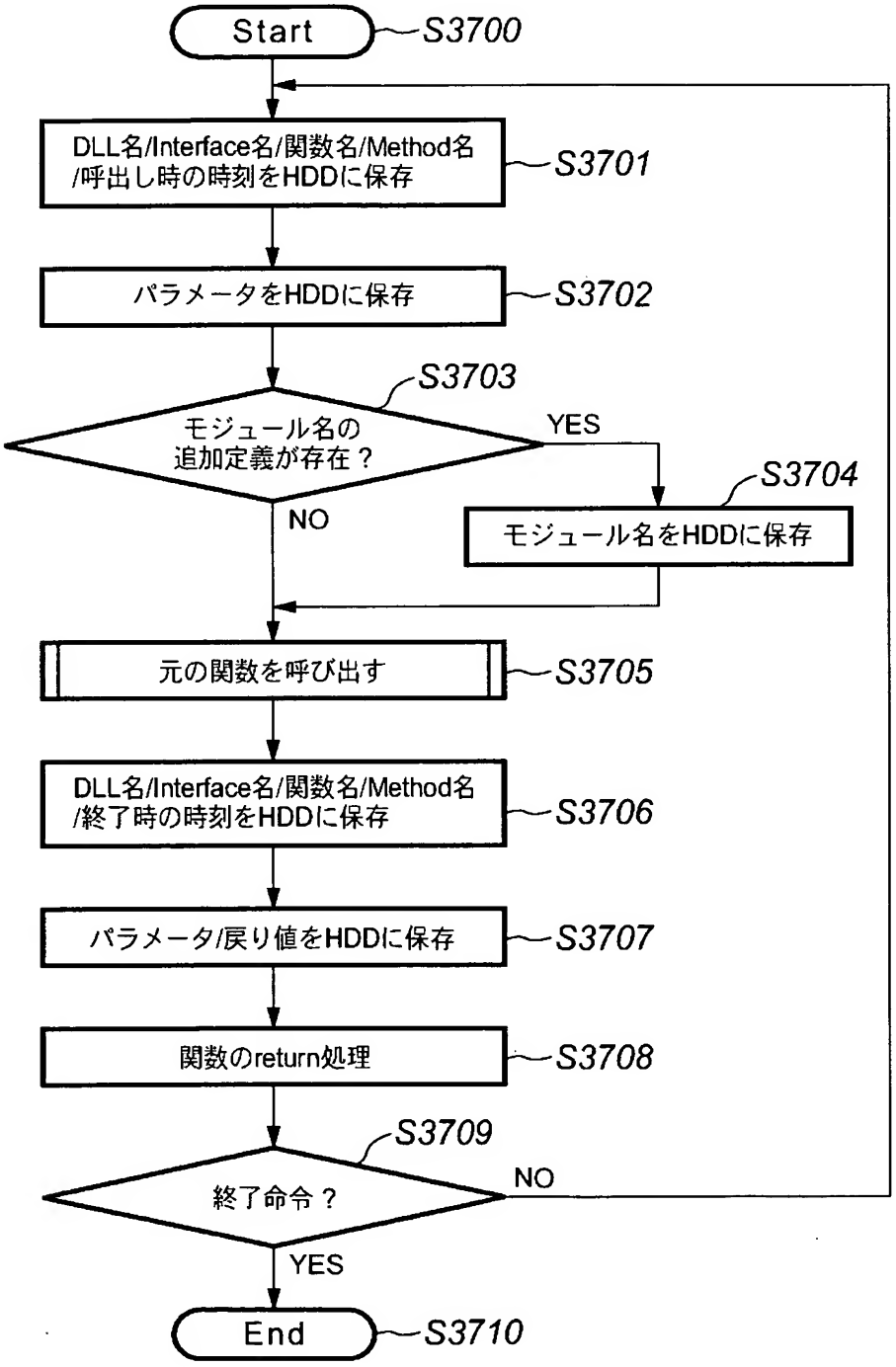
【図３５】

```
[Library]                    3500
LibraryName=TestDllStd                    3501
Patch=C:¥Windows¥System32¥spool¥drivers¥w32x86¥3
ModuleName=UnidrvUIPlugin.dll
...                          3502
```

【図３６】


[Interface]
3600
InterfaceName=test
3601
Patch=C: ¥Windows ¥System32 ¥spool ¥drivers ¥w32x86 ¥3
ModuleName=UnidrvUIPlugin. dll
...
3602

【図３７】


Start
S3700
DLL名/Interface名/関数名/Method名
/呼出し時の時刻をHDDに保存
S3701
パラメータをHDDに保存
S3702
S3703
モジュール名の
追加定義が存在？
YES
NO
S3704
モジュール名をHDDに保存
元の関数を呼び出す
S3705
DLL名/Interface名/関数名/Method名
/終了時の時刻をHDDに保存
S3706
パラメータ/戻り値をHDDに保存
S3707
関数のreturn処理
S3708
S3709
終了命令？
NO
YES
End
S3710

【書類名】　要約書

【要約】

【課題】　ソフトウェアの処理ログを容易に取得でき、かつ、バグの解析のための工数を削減することが可能なログ取得方法を提供する。

【解決手段】　関数を備えるプログラムのログ取得方法であって、プログラム実行時に呼び出されるＯＳ内の関数のうち、指定された関数を識別する工程と、ロードされた前記所定の処理を行う関数のアドレスと前記指定されたＯＳ内の関数のアドレスとを、ログ取得のための関数のアドレスに書き換える工程とを備え、前記ログ取得のための関数は、前記所定の処理を行う関数と前記指定されたＯＳ内の関数とを呼び出し、実行させた結果を前記プログラムに渡す工程（ステップＳ１７０８、Ｓ１７１２、Ｓ１７１３）と、前記所定の処理を行う関数と前記指定されたＯＳ内の関数とを呼び出す際および結果を受け取った際の所定の情報を記録する工程（ステップＳ１７０６、Ｓ１７０９）とを備える。

【選択図】　図１７

特願２００３－０９９４６５

出　願　人　履　歴　情　報

識別番号　［０００００１００７］

１．変更年月日　１９９０年　８月３０日
［変更理由］　新規登録
住　所　東京都大田区下丸子３丁目３０番２号
氏　名　キヤノン株式会社